Focused Random Walk with Probability Distribution for SAT with Long Clauses

Huimin Fu, Jun Liu and Yang Xu

Abstract

Focused random walk (FRW) is one of the most influential paradigm of stochastic local search (SLS) algorithms for the propositional satisfiability (SAT) problem. Recently, an interesting *probability distribution* (PD) strategy for variable selection was proposed and has been successfully used to improve SLS algorithms, resulting in state-of-the-art solvers. However, most solvers based on the PD strategy only use *polynomial function* (PoF) to handle the exponential decay and are still unsatisfactory in dealing with medium and huge *k*-SAT instances at and near the phase transition. The present paper is focused on handling all *k*-SAT instances with long clauses. Firstly, an extensive empirical study of one state-of-the-art FRW solver WalkSATlm on a wide range of SAT problems is presented with the focus given on fitting the distribution of the *break* value of variable selected in each step, which turns out to be a Boltzmann function. Using theses case studies as a basis, we propose a *pseudo normal function* (PNF) to fit the distribution of the *break* value of variable selected, which is actually a variation of the Boltzmann function. In addition, a new *tie-breaking flipping* (TBF) strategy is proposed to prevent the same variable from being flipped in consecutive steps. The PNF based PD strategy combined with the TBF strategy lead to a new variable selection heuristic named PNF-TBF. The PNF-TBF heuristic along with a *variable allocation value* (Vav) function are used to significantly improve ProbSAT, a state-of-the-art SLS solver, leads to a new FRW algorithm dubbed PNFSat, which achieves the state-of-the-art performance on a broad range of huge random 7-SAT instance near the phase transition as demonstrated via the extensive experimental studies. Some further improved versions on top of PNFSat are presented respectively, including PNFSat_alt, which achieves the state-of-the-art performance on the medium 7-SAT instances at the phase transition; PN&PoFSat, which achieves the state-of-the-art performance on a broad range of random 5-SAT benchmarks; as well as an integrated version of these three algorithms, named PDSat, which achieves the state-of-the-art performances on all huge and medium random *k*-SAT instances with long clauses as demonstrated via the comparative studies using different benchmarks.

Keywords: Probability Distribution · Satisfiability (SAT) · Focused random walk (FRW) · Stochastic local search (SLS)

1 Introduction

The propositional satisfiability (SAT) problem is one of the most widely studied NP-complete problems and plays an outstanding role in many domains of computer science and artificial intelligence due to its significant importance in both theory and applications [1]. Considering a propositional formula F in the Conjunctive Normal Form (CNF) defined on a set of Boolean variables, the SAT problem asks whether there exists a truth assignment to the variables of F that satisfies all clauses in F. The SAT problem is fundamental in solving many practical problems in combinatorial optimization, statistical physics, circuit verification, mathematical logic, machine learning, constraint satisfaction, real-time scheduling, and computing theory [2].

Since SAT solving is a practical domain, we need SAT instances to test different algorithms. The uniform random *k*-SAT instances are a well-studied category of SAT. The class of random *k*-SAT instances is a relatively unbiased sample for algorithms [3]. Random *k*-SAT instances remain very difficult. Indeed, such instances are challenging for all kinds of algorithms and by controlling the instance sizes and the clause-to-variable ratios, they provide adjustable hardness levels to assess the solving capabilities. Moreover, the performance of algorithm is usually stable on random *k*-SAT instances, either good or bad. Actually, the class of random *k*-SAT instances is one of the three main tracks in the well-known SAT competitions [4].

There are many optimization algorithms dedicated to different SAT solvers to solving SAT problems, which are divided into two main classes: one is complete, the other is incomplete. Complete algorithms are mainly based on DPLL [5, 6] and resolution principle [7]. The incomplete SAT solvers are mainly based on stochastic local search (SLS) algorithms which are among the best- known methods currently available

This work is supported by the National Natural Science Foundation of China (Grant No. 61673320) and the Fundamental Research Funds for the Central Universities (Grant No. 2682017ZT12, 2682016 CX119).

H. Fu is with the Key Laboratory of National-Local Joint Engineering Laboratory of System Credibility Automatic Verification of China, School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China (email: fhm6688@my.swjtu.edu.cn)

J. Liu is with the Key Laboratory of National-Local Joint Engineering Laboratory of System Credibility Automatic Verification of China, and also with the School of Computing, Ulster University, Northern Ireland, UK (email: j.liu@ulster.ac.uk)

Y. Xu is with the Key Laboratory of National-Local Joint Engineering Laboratory of System Credibility Automatic Verification of China, School of Mathematic, Southwest Jiaotong University, Chengdu, China (email: xuyang@swjtu.edu.cn)

for solving types of SAT problems. Although the incomplete SAT solvers cannot guarantee either to find the solutions or prove a given Boolean formula unsatisfiable, some of them are surprisingly more effective than state-of-the-art complete solvers on finding models of satisfiable formulae for random k-SAT instances [8]. SLS strategies can also be applied to solving traveling salesman problems by optimizing ant colony algorithm [9].

An SLS algorithm starts by generating randomly a truth assignment of the variables of F. Then it explores the search space to minimize the number of falsified clauses. To do this, it iteratively flips the truth value of a variable selected according to some heuristic at each step until it seeks out a solution or timeout. Hence, there are two main factors affecting SLS algorithms, one is to generate a complete initial assignment, and the other is a variable selection heuristic.

Among random k-SAT instances, random 3-SAT ones exhibit some particular statistical properties and are easy to solve, for example, by SLS algorithms and a statistical physics approach called *Survey Propagation* [37]. It has been shown that the famous SLS algorithm WalkSAT [36] scales linearly with the number of variables for random 3-SAT instances near the phase transition. The state-of-the-art FrwCB solves random 3-SAT instances near the phase transition (at ratio 4.2) with millions of variables within 2-3 hours [12].

However, random k-SAT instances with long clauses remain very difficult, and the performance of SLS algorithms on such instances has stagnated for a long time. Indeed, such instances are challenging for all kinds of algorithms, including the Survey Propagation algorithm, which solves random 3-SAT instances extremely fast [37]. Recently, a few progresses such as, CScoreSAT [32], ProbSAT [18] and YalSAT [10], have been made in this direction. In particular, when solving random instances near the phase transition, CScoreSAT is good at k-SAT with k>3, and ProbSAT is good at solving random 5-SAT and 7-SAT instances, and the YalSAT algorithm is good at solving random 5-SAT instances.

Most SLS solvers improve different variable selection heuristics to develop algorithms. Heuristics in SLS algorithms for SAT can be divided into two categories: two-mode SLS algorithms and focused random walk (FRW) algorithms. Recent solvers usually combine these two kinds of heuristics, such as the winners of random satisfiable category of SAT Competition 2017 and the silver award of random satisfiable category of SAT competition 2016 namely YalSAT [10] and CSCCSat [11].

FRW algorithms always select a variable to be flipped from an unsatisfied clause chosen randomly in each step [12]. On solving random k-SAT instances, FRW framework performs better than others. WalkSAT, regarded as the first FRW algorithm, firstly uses both noise factor and random walk strategy, then utilizes greedy strategy and still shows state-of-the-art performance in solving 3-SAT instances. WalkSATlm [13, 34] implemented several variants of WalkSAT's algorithm, and took a large step towards improving SLS algorithm for random k-SAT instances with k> 3. FrwCBlm [3] implemented a completely new configuration checking (CC) strategy based on clause states and showed great efficiency and robustness on random k-SAT instances with k>3.

Recently, one two-mode algorithm based on an interesting probability distribution (PD) strategy for variable selection had been proposed to handle the random k-SAT instances, resulting in an efficient two-mode algorithm, such as Sparrow [15], which is the winner of random satisfiable track of SAT competition 2011. Whereas previous heuristics select the flipping variable based on variables properties, the PD strategy takes the circumstance of the variable into account. The PD strategy for SAT in the literature [16] selects a variable x to be flipped by deciding whether it has the best *make* or the lowest *break* in an unsatisfied clause chosen randomly. Moreover, the experimental results in the literature [16] indicate that the FRW algorithms based on PD strategy dubbed ProbSATsc13, is more effective than two-model SLS algorithms and the winner of the random satisfiable track of SAT competition 2013. Afterwards, the PD heuristic has been further developed, such as polypower1.0 [17], which is the fourth place of random satisfiable track of SAT competition 2016; YalSAT [10], which implements several variants of ProbSAT's algorithm, and win the random satisfiable track of SAT competition 2017; ProbSAT [18], which is the second-ranked solver among the SLS solvers in terms of capability for the SAT competition 2018.

The literature [16] has showed that the exponential delay in probability with growing break value might be too strong in the case of 3-SAT, so the PD strategy selects a variable x to be flipped according to the polynomial function (PoF) of *break* value, and picks a variable x to be flipped according to the exponential function of *break* value for k-SAT with k>3. The FRW algorithm polypower1.0 [17] also deals with exponential decay in some sense, and it uses a PoF to solve random k-SAT instances. However, there are some limitations in previous FRW algorithms based on the PD strategy, which only use PoF to handle the exponential decay for random k-SAT instances, and thus lose their power, especially for solving random k-SAT instances with k>3. Empirical evidences, which present the ineffectiveness of the PD strategy only based on the PoF, can be found in Section 6. In this paper, we propose a new fitting function strategy that works much better on the problem of exponential delay.

The first contribution of the present work is summarized below: we use an internationally renowned FRW algorithm WalkSATlm [13] to test all medium and huge random k-SAT instances from SAT competition 2017 and 2018, with the aim to fit the distribution of the average ratio of the total times of variables corresponding to each *break* value in all variables selected and the total times of variables corresponding to each *break* value in all randomly unsatisfied clauses selected in the solution process for all random k-SAT instances that can be solved by WalkSATlm, while the fitting function is consistent with the so-called Boltzmann function. Since WalkSATlm utilizes the noise strategy, there is a certain probability that the variables are randomly selected. We found that the smaller the probability in the noise strategy is, the smaller the error between the distribution of the ratio of *break* value and the

Boltzmann function is.

The second contribution of the present work is to adapt the fitting function to make it applicable to FRW algorithms. A new alternative PD strategy based on a new probability function, called *pseudo normal function* (PNF), is proposed. Then we propose a new variable selection heuristic, called PNF-TBF, which combines the PNF and a new tie-breaking flipping (TBF) strategy in a subtle way. Then we combine PNF-TBF with the recently proposed *variable allocation value* (Vav) function [20], resulting in a new FRW algorithm named PNFSat. The experiments show that PNFSat exhibits the best performance on huge random 7-SAT instances. Further analysis for PNFSat indicates that the new tie-breaking strategy is not suitable for solving medium 7-SAT instances at the threshold ratio of the solubility phase transition, but the alternation version PNFSat_alt (PNFSat without the new tie-breaking strategy) significantly outperforms its FRW competitors (which are based on the PD strategy), namely ProbSAT [18] and YalSAT [10] as well as the currently best two-mode SLS solver Score_2SAT [19] on such instances.

The third contribution of the present work is to improve the performance of PNFSat on solving medium and huge random 5-SAT instances with various ratios and sizes. Based on the Boltzmann function, we propose two new variable selection heuristics called PN-PoF and Po-PNF respectively, which reflects a combined use of both PN-PoF and Po-PNF on top of PNFSat, leading to a new FRW algorithm dubbed PN&PoFSat. Significantly improving PNFSat, PN&PoFSat achieves state-of-the-art performance on random 5-SAT instances. Furthermore, our experiments show that PN&PoFSat exhibits the best performance on huge and medium random 5-SAT instances in terms of total success runs.

Additionally, the fourth contribution of the present work is that we combine PNFSat, PNFSat_alt and PN&PoFSat, leading to a new flexible FRW algorithms called PDSat. Our evaluations present that PDSat dramatically outperforms state-of-the-art SLS solvers on all huge and medium random *k*-SAT instances with long clauses, including FRW algorithms namely WalkSATlm, YalSAT and ProbSAT, and two-mode SLS algorithms namely Sparrow [15], DCCASat [21], CSCCSat [11], and Score_2SAT.

Finally, we provide discussions about the implementation of the PDSat algorithm in our work, and do further empirical analyses on comparing PoF, PNF, PN-PoF and Po-PNF, *Vav* function, and the new TBF mechanism. According to our observations, PoF loses its effectiveness when applying to the problem of exponential decay on random *k*-SAT instances with long clauses, and to the best of our knowledge, PDSat is currently the only PD strategy that can be used to improve the problem of exponential decay and the performance of PD strategy based FRW algorithms.

This paper is structured as follows. In Section 2, we provide some necessary preliminaries. Section 3 discusses the fitting function of the *break* value of variables selected in WalkSATlm, i.e., a Boltzmann function. In Section 4, we propose the new PNF distribution of the *break* value of variable selected, which is actually a variation of the Boltzmann function. In Section 5, a new TBF strategy is proposed, followed by the PNF-TBF heuristic based on the PNF and TBF, which led to a new FRW algorithm called PNFSat, an alternative version of PNFSat, called PNFSat_alt is also provided, their performances are demonstrated with the detailed experimental studies. In Section 6, we propose the Po-PNF heuristic and PN-PoF heuristic, and introduce the PN&PoFSat algorithm which reflects a combined use of the above two heuristics on top of PNFSat. The empirical results of PN&PoFSat are also provided. Section 7 discusses the integrated algorithm of PNFSat, PNFSat_alt and PN&PoFSat, called PDSat, along with its experimental evaluation. Further discussions on the approximate implementation of PDSat and empirical analyzes on PNF, Po-PNF, PN-PoF, PoF, *Vav* function and the new TFB scheme applied to FRW algorithms are demonstrated in Section 8. Finally, Section 9 concludes the paper and lists some future work.

2 Preliminaries

A formula F of the SAT is defined by a pair $F=(X, C)$ such that $X=\{x_1, x_2,..., x_n\}$ is a set of n Boolean variables (their values belong to the set {true, false}) and $C=\{c_1, c_2, ..., c_n\}$ is a set of m clauses. A clause $c_i \in C$ is a disjunction of literals and a literal is either a variable x_i (which is called positive literal) or its negation $\neg x_i$ (which is called negative literal). We define $C(x)=\{c \mid c$ is a clause which x appears in$\}$. A clause can also be represented by the set of its literals. For a set of literals L, $var(L)$ is the set of the variables in L. Accordingly, $var(c_i)$ is the set containing the variables appearing in c_i. The size of a clause c_i is the number of its literals and it is denoted by $|c_i| = |var(c_i)|$. If the size of each clause in C is equal to k ($\forall c_i \in C$, $|c_i|=k$) then the instance is a k–SAT instance and $r = m/n$ is its clause-to-variable ratio. An instance $F=c_1\wedge c_2\wedge...\wedge c_m$ is a conjunction of clauses.

A satisfying assignment α for a formula F is an assignment to its variables such that the formula evaluates to true. If x_i is true by α then x_i belongs to α (otherwise $\neg x_i \in \alpha$). A set of all unsatisfied clauses under a complete assignment a for a formula F is defined by $unsat(\alpha)$. Given an instance F, the SAT problem is to find a satisfying assignment or prove that none exists. A literal l is said to be satisfied by the current value of the variable α if $l \in \alpha$ and falsified if $\neg l \in \alpha$. A clause is satisfied by α if at least one of its literals is true literal and falsified otherwise. A clause is t-satisfied if and only if it includes exactly t true literals under α [13]. A solution of F is an assignment that satisfies all the clauses of F.

The SLS algorithm generally generates a random complete assignment. Recently, the v*ariable allocation value* (Vav) function [31] is proposed to generate a greedy initial assignment. The *Vav* function of a variable x is the number of occurrences of literal x divided by the number of occurrences of literal $\neg x$. If the *Vav* function of the variable x is greater than the specified parameter, the initial assignment of the variable x is true; if the *Vav* function of the variable x is less than another specified parameter, the initial the initial assignment of the variable x is false; otherwise the initial assignment of the variable x is assigned randomly.

In each step, the mainly variable x properties used by SLS algorithms for SAT are *make*(x) [13], [22] and *break*(x) [23], which are the number of clauses that would become satisfied and unsatisfied respectively, if variables x were to be flipped. Usually, SLS algorithms for random k-SAT instances select a variable x to be flipped based on its properties of *score*(x) [24], [25], [26] and *age*(x) [27]. A scoring function which can be a simple property or any mathematical expression with one or more properties measures the increase in the number of satisfied clauses by flipping x, and *score*(x) is defined as *make*(x)–*break*(x). *age*(x) is defined as the number of steps since the variable x was last flipped.

2.1 Probability distribution strategy review

Probability distribution (PD) techniques have proven successful in FRW algorithms [10], [15], [17], [18]. PD strategy is based on the PoF, which aims to handle the cycling problem in local search [18]. When algorithms based on PD strategy reached a local minimum, they compute a PD on the variables from an unsatisfied clause. In the context of SAT, originally state in the literature [15], given a formula F and a complete assignment α, the probability distribution of a variable x takes into account the difference between the *score*(x) and the *age*(x) of variables.

In addition to FRW SLS algorithms based on PD strategy, previous SLS algorithms for SAT always utilize the greedy strategy. They usually select the flipping variable according to the properties of variables x, such as *make*(x), *break*(x), *score*(x), and *circumstance information* [28], [29], [30]. Greedy strategy is easy to fall into the local minimum. However, compare with other state-of-the-art SLS algorithms, PD based ones need neither noise nor a random walk or greedy strategy to escape efficiently from cycles. The PD strategy is a simple and efficient method [15]. This is the essential difference between the PD strategy and previous works.

2.2 ProbSAT review

In this section, we briefly review the ProbSAT algorithm [18], which serves as the basic of our proposed algorithms in the later sections. The ProbSAT algorithm is a recent milestone in local search for solving SAT. Just after it was proposed, it becomes the basic framework of Dimetheus and YalSAT. Yalsat won the random track of SAT Competition 2017, Dimetheus won the RSC 2014 and 2016.

The PD strategy used in ProbSAT is based on a polynomial function or an exponential shape, $f(x, \alpha)$, as listed below respectively:

$$f(x,\alpha) = \left(\mathcal{E} + break(x,\alpha)\right)^{-c_b} \quad (1)$$

or

$$f(x,\alpha) = (c_b)^{-break(x,\alpha)} \quad (2)$$

where c_b and $\mathcal{E}$ are two parameters.

The pseudo-code of ProbSAT is described in Algorithm 1 and can be found in the literature [18].

Agorithm 1: ProbSAT algorithm

Input: CNF-formula F, *MaxTries*, *MaxSteps*
Output: A satisfying assignment α of F, or "UNKNOWN"
1 **begin**
2   **for** i = 1 to *MaxTries* **do**
3     $\alpha \leftarrow$a generated truth assignment randomly for F;
4   **for** j = 1 to *MaxSteps* **do**
5   **if** α *satisfies* F **then Return** α;
6       $C \leftarrow$an unsatisfied clause chosen at random;
7   **for** x in C **do**
8   compute $f(x, \alpha)$;
9         $x \leftarrow$random variable x according to probability $\frac{f(x,\alpha)}{\sum_{z \in C} f(z,\alpha)}$;
10   **end for**
11       $\alpha \leftarrow \alpha$ with x flipped;
12   **end for**
13   **end for**
14   **Return** "UNKNOWN";
15 **end**

In the beginning, ProbSAT algorithm performs the first loop until it finds a satisfying assignment or reaches the first limited steps denoted by *MaxTries*. Then ProbSAT algorithm generates a complete assignment α randomly as the initial assignment (line 3 in Algorithm 1). Then ProbSAT algorithm starts the second loop until a satisfying solution is found or reaches the second limited steps denoted by *MaxSteps*. During the search process, ProbSAT algorithm selects an unsatisfied clause randomly (line 6 in Algorithm 1), and then ProbSAT tries to select a flipping variable based on probability (line 7-10in Algorithm1) to be flipped (line 11 in Algorithm 1). Finally, once the search process terminates, the ProbSAT reports α as the solution; otherwise, ProbSAT reports UNKNOWN.

ProbSAT algorithm explores the search space to minimize the number of unsatisfied clauses. To do this, it is natural for ProbSAT algorithm to select a variable to be flipped.

3 Boltzmann fitting function of break value in WalkSATlm

WalkSATlm is a typical and state-of-the-art FRW algorithm, which utilizes greedy strategy and random walk. As WalkSATlm and PD strategy based algorithms are two completely different FRW algorithms, *a natural question is whether there exists an alternative function which can reflect the dynamics of the variable in the solution process of WalkSATlm, and can also be used to guide the solution of the PD strategy based algorithms.* The literature [16] has shown that *break* value is the best important factor and PD strategy can even do without the *make* value completely. Hence, in this section we only fit the distribution of *break* value of variables selected by WalkSATlm.

3.1 Experiment preliminaries

To test WalkSATlm, we set up four benchmarks:

1) **7-SAT_huge**: The benchmark contains all 40 huge random 7-SAT instances from SAT Competition 2017[1] and

[1]https://baldur.iti.kit.edu/sat-competition-2017/benchmarks/

SAT Competition 2018[2] ($16.0 \le r \le 19.8$, n= 250000, two instances each size).

2) **7-SAT_medium:** The benchmark includes all 50 medium random 7-SAT instances ($r = 87.79$, 90≤n≤ 168, one instance each size except for 11 instances of n=120) from SAT Competitions 2017 and 2018.

3) **5-SAT_huge**: The benchmark contains all 40 huge random 5-SAT instances ($16.0 \le r \le 19.8$, n=250000, two instances each size) from SAT Competition 2017 and SAT Competition 2018.

4) **5-SAT_medium**: The benchmark includes all 50 medium random 5-SAT instances ($r = 21.117$, 220≤n≤ 590, one instance each size except for 11 instances of n=250) from SAT Competitions 2017 and 2018.

The binary of WalkSATlm is downloaded from the webpage of SAT Competition 2016[3].

In this paper, all experiments are carried out on a machine under a 64-bit Ubuntu Linux Operation System, using 2 cores of Intel(R) Core (TM) i3-3240M 3.4 GHz CPU. WalkSATlm is performed for ten runs on each instance within 2000s.

3.2 Fitting function of *break* value in WalkSATlm

In this section, we fit the distribution of the average ratio of the total times of variables corresponding to each *break* value in all variables selected and the total times of variables corresponding to each *break* value in all randomly unsatisfied clauses selected in the solution process for all random k-SAT instances with long clauses that can be solved by WalkSATlm, and the results are shown in Figs. 1-4. The boxes in Figs. 1-4 are the detailed information of the fitting function. When the *break* value is greater than 19, the average ratio is 0, so we have done a uniform sampling of *break* values belong to [0, 19].

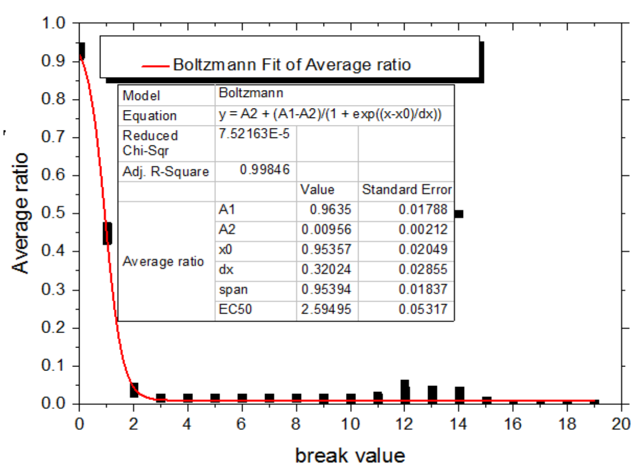

Fig. 1. Average ratio of the average times of break value in all variables selected and the average times of break value in all randomly unsatisfied clauses selected. All black squares refer to the discrete distribution of the average break value ratio of all 7-SAT_huge benchmark solved within the time limit. The curve is the fitted distribution of all discrete points.

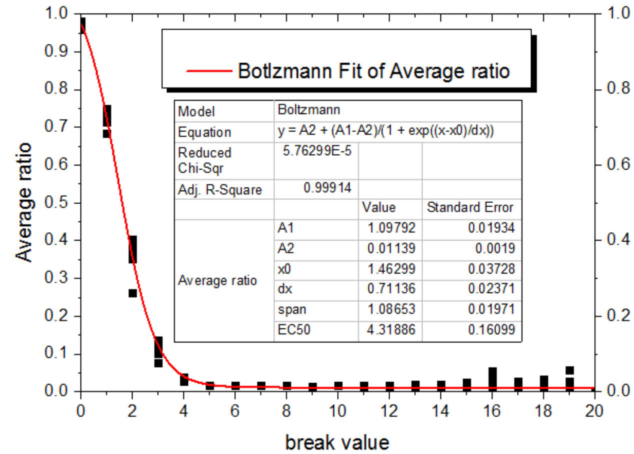

Fig. 2. Average ratio of the average times of break value in all variables selected and the average times of break value in all randomly unsatisfied clauses selected. All black squares refer to the discrete distribution of the average break value ratio of all 7-SAT_medium benchmark solved. The curve is the fitted distribution of all discrete points.

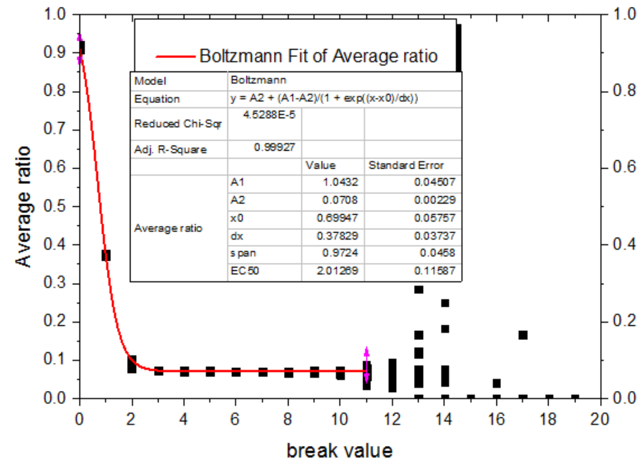

Fig. 3. Average ratio of the average times of break value in all variables selected and the average times of break value in all randomly unsatisfied clauses selected. All black squares refer to the discrete distribution of the average break value ratio of all 5-SAT_huge benchmark solved. The curve is the fitted distribution for all discrete points with break value from 0 to 11.

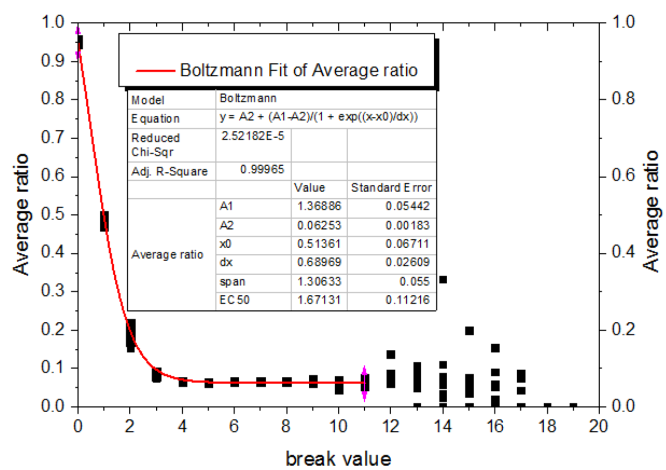

Fig. 4. Average ratio of the average times of break value in all variables selected and the average times of break value in all randomly unsatisfied clauses selected. All black squares refer to the discrete distribution of the average break value ratio of all 5-SAT_medium benchmark solved. The curve is the fitted distribution for all discrete points with break value from 0 to 11.

[2]http://sat2018.forsyte.tuwien.ac.at/
[3]https://baldur.iti.kit.edu/sat-competition-2016/

Result summary: In the case of 7-SAT_huge benchmark and 7-SAT_medium benchmark, a very good fit function which is a Boltzmann function has been obtained, and the error of the discrete points and the fitting function is very small. In the case of 5-SAT_huge benchmark and 5-SAT_medium benchmark, when the *break* value is less than 12, there is a good fitting function, which also satisfies a Boltzmann function in addition to the different parameter settings.

WalkSATlm utilizes random walk strategy by a noise factor *wp*. In fact, *wp* is 0.390 on random 5-SAT_huge benchmark; *wp* is 0.351 on random 5-SAT _medium benchmark; *wp* is 0.120 on random 7-SAT_huge benchmark; *wp* is 0.115 on random 7-SAT _medium benchmark in WalkSATlm algorithm. Hence, when the *break* value is greater than 11, the bigger the *wp* is, the more variables are randomly selected to flip. So, we just fit the function for the discrete points with the *break* values from 0 to 11 on random 5-SAT_huge benchmark and 5-SAT_medium benchmark.

Hence, when the *break* value is from 0 to 19, if *wp* approaches to 0, the distribution of between the *break* value and the utilization rate of *break* value, which is the average ratio of the total times of variables corresponding to each *break* value in all variables selected and the total times of variables corresponding to each *break* value in all randomly unsatisfied clauses selected in the solution process for all random *k*-SAT instances that can be solved by WalkSATlm, tends to be a Boltzmann function.

4 Pseudo normal function as a new probability function

As PD strategy is still in its infancy for solving SAT problem, a natural question is whether there exists an alternative PD strategy for SAT which is more efficient and robust than PoF based PD strategy on improving SLS algorithms for *k*-SAT instances with long clauses. In this section, we propose a novel probability function, called a *pseudo normal function* (PNF), which is actually a variation of the Boltzmann function, which forms a basis for an alternative PD strategy.

It has turned out that the influence of *break* is rather strong [16]. Thus, it is reasonable if we only consider the *break* value completely, it still leads to very good algorithms, while its implementation is simple and has less overhead. On the other hand, note that the Boltzmann function reflects the probability distribution of the *break* value when WalkSATlm solves the random *k*-SAT instances with long clauses, and is close to the exponential distribution. The above considerations suggest two principles in designing the new alternative PD strategy as below:

1) The *break* value property plays a very important role;

2) When WalkSATlm solves the random *k*-SAT instances with long clauses, the probability distribution of the *break* value is close to exponential distribution.

As a result, the normal function of the *break* value is defined firstly, followed by a new notion of pseudo normal function.

***Definition* 1**: For a CNF formula *F*, a complete assignment α to *var*(*F*), the normal function, denoted by *NF*, is a function on *var*(*F*) such that

$$NF(x,\alpha) = \frac{1}{\sqrt{2*\pi}} * e^{-\frac{break(x,\alpha)^2}{2}} \tag{3}$$

***Definition* 2**: For a CNF formula *F*, a complete assignment α to *var*(*F*), the pseudo normal function, denoted by PNF, is a function on *var*(*F*) such that

$$PNF(x,\alpha) = \pi * \frac{1}{\sqrt{2*\pi}} * e^{-\frac{break(x,\alpha)^2}{2}} = \pi * NF(x,\alpha) \tag{4}$$

PNF is an exponential function of the *break* value. This function is so simple and can be computed with little overhead.

Note that PNF is different from the exponential function utilized in ProbSAT [18]. Especially, the exponential function of ProbSAT is based on the *break* value, while PNF is based on the square of the *break* value. Moreover, ProbSAT has parameter involved, while there is no parameter in the PNF.

5 Proposed algorithm PNFSat for random 7-SAT

In this section, we introduce the main ideas in the proposed algorithm for random 7-SAT. We firstly apply a new PD strategy based on PNF to the FRW SLS paradigm and then present a new tie-breaking strategy, both of which leads to a new variable selection heuristic, called a PNF-TBF heuristic.

5.1 Applying the PNF to the FRW SLS paradigm

In this section, we apply the PNF to the focused random walking (FRW) SLS paradigm.

Previous PD strategy based SLS algorithm for random 7-SAT problems can be categorized into two classes: (i) some early PD strategy based on PoF for random 7-SAT [17]; (ii) recent studies, mainly including ProbSAT [18] and YalSAT [10] as well as their variants, used the PD strategy based on the exponential function for random 7-SAT. The exponential function turns out to be more effective than PoF for solving 7-SAT problems. In the present work, we propose to utilize the PD strategy based on the PNF described in Section 4 to identify a “good” variable which has the minimum *break* value.

5.2 The new tie-breaking strategy

Currently, there are two most popular variable selection strategies for solving SAT: probability function and configuration checking (CC) strategy [30].

Adopting the probability based on the PNF to pick a variable to be flipped may select the same variable in consecutive steps, so that it makes useless search in consecutive steps. Therefore, it is expected that the last flipping variable could not be the current flipping variable based on the idea of configuration checking (CC) strategy, which has proved to be effective in SLS algorithms for solving SAT [11, 19]. Thus, it is reasonable for us to employ a tie-breaking strategy that avoids selection of the same variable in consecutive steps.

The proposed tie-breaking strategy is inspired by the idea in the literature [13], but they are essentially different from each other. The latter may not be suitable for algorithms based on the PD strategy. The main difference lines in that in our proposal, a variable is mainly selected based on the PNF, there is no need to select one from all those variables with the same minimum break value in the selected clause.

Before introducing the new tie-breaking strategy, we introduce the *tie-breaking flipping* (TBF) variable firstly.

Definition 3: For a CNF formula F, a complete assignment α to $var(F)$, and the last flipped variable y, for a set of all unsatisfied clauses $unsat(\alpha)$, a variable x is *tie-breaking flipping* (TBF) if and only if $x \in var(unsat(\alpha))$ and $x \neq y$.

Note that a variable x is a TBF of unsatisfied clause c if and only if $x \in var(c)$ and $x \neq y$.

In this paper, we use $TBFVar(\alpha)$ to denote the set of all TBF variables of F under α. The $TBFVar(c)$ is denoted the set of all TBF variables of an unsatisfied clause c.

The new tie-breaking strategy, called TBF, is described as follows:

- When the TBF strategy is called, if there exists a variable selected by the PNF based PD strategy which is the same as the last flipped variable y, then if the number of unsatisfied clause is less than parameter R, it prefers to pick a **TBF variable of clause *c*** as the flipping variable randomly;

- Otherwise, the TBF strategy prefers to randomly pick a **TBF variable of *F*** as the flipping variable, leading the algorithm to search deeply, and preventing the algorithm from revisiting the recently faced scenario.

5.3 The PNF-TBF heuristic

According to the PNF and the new tie-breaking strategy, we design a new variable selection heuristic named PNF-TBF.

Specially, the PNF-TBF heuristic works as follows. After randomly selecting an unsatisfied clause c, PNF-TBF switches between two levels, i.e., the probability level and TBF level, depending on whether the variable x selected by PD strategy based on PNF is the same as the last flipped variable y or not. If $x \neq y$, PNF-TBF works in the probability level; otherwise it works in the TBF level. In probability level, PNF-TBF prefers to choose the variable x with the minimum $break(x)$ in the clause c. In the TBF level, if the number of unsatisfied clauses ($numFalse$) is less than the parameter R, PNF-TBF chooses the variable x randomly in $TBFVar(c)$; otherwise, it chooses the variable x randomly in $TBFVar(\alpha)$.

In brief, the new variable selection mechanism based on PNF-TBF heuristic is achieved by selecting the variables by probability based on PNF; once ties occur, a new tie-breaking strategy breaks ties of variables and selects a variable by the new tie-breaking strategy.

Algorithm 2: PNFSat (F)

```
Input: CNF-formula F, MaxTries, MaxSteps
Output: A satisfying assignment α of F, or UNKNOWN
1  for i = 1 to MaxTries do
2      α ←a generated assignment for F by variable allocation value function;
3      for j = 1 to MaxSteps do
4          if α satisfies F then
5              return α;
6          C ←an unsatisfied clause selected randomly;
7          for x in C do
8              f(x, α) ←compute PNF(x, α);
9          v ←random variable x according to probability f(x,α) / Σ_{z∈C} f(z,α);
10         if ( v==bestVar ) then
11             if ( numFalse<R) then
12                 bestVar ←random variable x in TBFVar(c);
13             else
14                 bestVar ←random variable x in TBFVar(α);
15         else
16             bestVar=v;
17         flip(bestVar);
18 return UNKNOWN;
```

5.4 PNFSat algorithm

In this section, we utilize the PNF-TBF heuristic and the *variable allocation value* (Vav) function [20] to improve ProbSAT algorithm and make a serious modification on ProbSAT, resulting in a new FRW algorithm dubbed PNFSat. The pseudo-code of the PNFSat algorithm is outlined in Algorithm 2. Before getting into the details of the PNFSat algorithm, we first introduce two modifications employed in the algorithm.

PNFSat differs from ProbSAT in the following two aspects. Firstly, although both algorithms utilize the PD strategy, the PD strategy in PNFSat is based on the PNF for solving random 7-SAT, while ProbSAT use the PD strategy based on the exponential function described in Section 2.2. Secondly, PNFSat utilizes the new tie-breaking to break ties, while ProbSAT does not use any tie-breaking strategies.

Initially, PNFSat performs the first loop until it finds a satisfying assignment or reaches the first limited steps denoted by *MaxSteps*. Then PNFSat generates a complete assignment α by a Vav function as the initial solution. Then it executes the second loop until a solution is found or reaches the second limited steps denoted by *MaxTries*.

In each search step, PNFSat picks a variable to be flipped. It performs the random walk to select an unsatisfied (line 6 in Algorithm 2), and then picks a variable according to the PNF-based PD strategy which is presented in Section 4 (lines 7-9 in Algorithm 2) and the new tie-breaking strategy (lines (10-16 in Algorithm 2): PNFSat first picks a variable by the PNF-based PD strategy, and then if the variable is the same as the last flipped variable and the number of unsatisfied clause is less than parameter R, PNFSat prefers to pick a TBF variable of clause c to be flipped randomly (lines 11 and 12 in Algorithm 2); otherwise, the new tie-breaking prefers to randomly pick a TBF variable of F to be flipped (lines 13 and 14 in Algorithm 2). After the variable to be flipped is selected, the PNFSat flips the selected variable (line 17 in Algorithm 2), then the PNFSat algorithm starts the next search step.

Finally, when the search terminates, if α satisfies all clauses of F, PNFSat outputs α as the solution; otherwise, PNFSat reports UNKNOWN.

5.4.1 Experimental preliminaries of PNFSat

In this subsection, we evaluate PNFSat on extensive random 7-SAT instances. Some experiment preliminaries are given below first.

Benchmarks: All the instances used in the following experiments are generated according to the random k-SAT model [35]. we adopt the following five benchmarks for uniform random 7-SAT.

1) **7-SAT_huge_SC18**: The benchmark includes all 20 huge random 7-SAT instances with $55 \le r \le 59$, $60 \le r \le 64$, $65 \le r \le 69$, $70 \le r \le 74$ (n= 50000, five instances each group) from SAT Competition 2018.

2) **7-SAT_huge_SC17**: The benchmark contains all huge 7-SAT instances with $55 \le r \le 59$, $60 \le r \le 64$, $65 \le r \le 69$, $70 \le r \le 74$ (n= 50000, five instances each group) from SAT Competition 2017.

3) **7-SAT _huge_SC16**: The benchmark consists of all 20 huge random 7-SAT instances with various ratio ($55 \le r \le 74$, n=50000, one instance each ratio) from SAT Competition 2016.

4) **7-SAT_huge:** Random 7-SAT instances near the threshold ratio of phase transition, generated random by the random k-SAT model (r=66.0, n=50000, 20 instances).

5) **7-SAT_medium _SC**: The benchmark contains all 90 medium random 7-SAT instances with $90 \le n \le 108$, $110 \le n \le 128$, $130 \le n \le 148$ and $150 \le n \le 168$ (r= 87.79, 30 instances expect for the second group, 20 instances the other group) from SAT Competition 2016 , 2017 and 2018.

The medium k-SAT benchmarks have a *clause-to-variable* ratio equal to the conjectured threshold ratio of the solubility phase transition, for which 50% of the uniform k-SAT instances are satisfiable, and a significant traction (about 50%) of the medium k-SAT instances is unsatisfiable. For most algorithms, instances generated closer to the phase-transition ratio are harder to solve [4].

The PNFSat algorithm is implemented in C/C++. For the parameters of the Vav function in PNASat, we utilize the default parameter setting tuned in the literature [20]. The parameter R for the TBF strategy in PNFSat are tuned according to our experience, and the parameter setting is R=3.

We compare PNFSat with four state-of-the-art FRW solvers, including WalkSATlm [13], FrwCBlm [3], YalSAT [10] and ProbSAT [18], and one state-of-the-art two-mode solver Score$_2$SAT. WalkSATlm and FrwCBlm are still highly competitive with the state-of-the-art on random k-SAT instances with long clauses. YalSAT is the winner of the random track of SAT Competition 2017. ProbSAT wins the gold medal of the random SAT track in SAT Competition 2013, is the second-ranked solver among the SLS solvers in terms of capability for the SAT competition 2018 and the current best FRW solver. Score$_2$SAT wins the bronze of SAT Competition 2017, but its performance outperforms the winner of SAT Competition 2017 on random k-SAT instances with long clauses[4] and it is the best two-mode SLS solver on random k-SAT instances.

In this paper, for WalkSATlm and FrwCBlm the parameters are set as the ones used in SAT Competition 2016. The source code of FrwCBlm can be downloaded online [5]. The YalSAT and Score$_2$SAT solvers we adopt are the two submitted to SAT Competition 2017 [6] . The binary of ProbSAT can be downloaded online[7] and we use the parameter setting as the one used in SAT Competition 2018.

In this paper, for each solver on each instance group, we report the number of success runs (#suc), as well as “par 10”, which is a penalized average run time where an unsuccessful run of a solver is penalized as 10 times cutoff time, and “Over all” symbols averaged over all instances with each run per instance. Note that PAR 10 is adopted in SAT Competitions and has been widely used in the literature as a prominent performance measure for SLS solvers [32]. If a solver has no successful run on an instance class, the corresponding “par10” is marked with “n/a”.

For the 7-SAT_huge_SC17 benchmark and 7-SAT_huge_ SC18 benchmark, each solver is performed for twenty runs for each instance. For the 7-SAT_huge_SC16 benchmark and 7-SAT_medium_SC benchmark as well as 7-SAT_huge benchmark, each solver is performed for five runs for each instance. The cutoff time for all runs is set to 5000 seconds as same as SAT competitions in 2016, 2017 and 2018.

5.4.2 Experiment results for PNFSat

1) ***Results on the* 7*-SAT_huge_SC*18 *benchmark**:* Table 1 illustrates comparative results of PNFSat and its competitors on the huge random 7-SAT benchmark from SAT Competition 2018. None of the solvers can solve any huge 7-SAT instances with ratios between 70 and 74, indicating that random 7-SAT instances near the phase transition are so difficult. Nevertheless, PNFSat shows significant superiority over ProbSAT and performs much better than the other competitors on the whole benchmark. Especially, on the random 7-SAT instance with 65≤r≤69, PNFSat is the only solver that solves 20 runs. Actually, all the competitors become ineffective (among which WalkSATlm, FrwCBlm and ProbSAT have the highest success rate of 20%) on the random 7SAT instance with65≤r≤69, whereas PNASat still achieves a success rate of 40% for this instance class. Also, PNFSat significantly outperforms its competitors in terms of run time, which is more obvious as the instances ratio increases.

2) ***Results on the* 7*-SAT_huge_SC*17 *benchmark**:* To show the robustness of PNFSat, we compare PNFSat with its competitors on the huge random 7-SAT benchmark from SAT Competition 2017. Table 2 presents the results of PNFSat and its competitors on this benchmark. According to the results, PNFSat stands out the best solver and significantly performs better than its competitors, which confirms the robustness of PNFSat on huge random 7-SAT instances.

[4]https://baldur.iti.kit.edu/sat-competition-2017/index.php?cat=results

[5]https://baldur.iti.kit.edu/sat-competition-2016/solvers/

[6]https://baldur.iti.kit.edu/sat-competition-2017/solvers/

[7]http://sat2018.forsyte.tuwien.ac.at/solvers/random/

Table 1: Comparative result of PNFSat and its competitors on the 7-SAT_huge_SC18

Instance Class	WalkSATlm		FrwCBlm		ProbSAT		Yalsat		PNFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
55≤*r*≤59	50	19	50	14	50	9	50	21	**50**	**8**
60≤*r*≤64	50	432	50	395	50	20	40	10062	**50**	**15**
65≤*r*≤69	10	40433	10	40521	10	40058	0	n/a	**20**	**30079**
70≤*r*≤74	0	n/a	0	n/a	0	n/a	0	n/a	**0**	**n/a**
Over all	110	22721	110	22732	110	22522	90	27521	**120**	**20026**

Table 2: Comparative result of PNFSat and its competitors on the 7-SAT_huge_SC17

Instance Class	WalkSATlm		FrwCBlm		ProbSAT		Yalsat		PNFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
55≤*r*≤59	50	21	50	16	50	9	50	21	**50**	**8**
60≤*r*≤64	50	406	50	474	50	24	40	10236	**50**	**16**
65≤*r*≤69	10	40586	10	40621	10	40021	0	n/a	**20**	**32092**
70≤*r*≤74	0	n/a	0	n/a	0	n/a	0	n/a	**0**	**n/a**
Over all	110	22754	110	22778	110	22514	90	27517	**120**	**22220**

Table 3: Comparative result of PNFSat and its competitors on the 7-SAT_huge_SC16

#Total runs	WalkSATlm		FrwCBlm		ProbSAT		Yalsat		PNFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
100	55	22729	55	22776	55	22512	50	26430	**55**	**22508**

Table 4: Comparative result of PNFSat and its competitors on the 7-SAT_huge

#Total runs	WalkSATlm		FrwCBlm		ProbSAT		Yalsat		PNFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
100	0	n/a	0	n/a	0	n/a	0	n/a	**65**	**18368**

Table 5: Comparative result of PNFSat and its competitors on the 7-SAT_medium_SC

Instance Class	WalkSATlm		FrwCBlm		ProbSAT		Yalsat		Score$_2$SAT		PNFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
90≤*n*≤108	70	15345	65	17532	70	15056	70	15063	70	15077	**70**	**15028**
110≤*n*≤128	80	23791	80	23656	**90**	**20584**	75	25264	85	22044	85	22116
130≤*n*≤148	**55**	**23558**	35	33065	45	27984	40	30463	45	27812	45	28203
150≤*n*≤168	30	**35369**	10	45204	20	40187	15	42701	30	35507	10	45178
Over all	235	24435	190	29174	225	25359	200	28028	230	24769	210	27044

3) ***Results on the 7-SAT_huge_SC16 benchmark****:* To solve the huge random 7-SAT instances, Table 3 presents the results of PNFSat and its competitors on this benchmark. Although PNFSat, ProbSAT, WalkSATlm and FrwCBlm succeed in 55 runs, the average time of PNFSat is 22508 s, whereas those of ProbSAT, WalkSATlm and FrwCBlm are 22512, 22729 and 22776, respectively. Also, YalSAT succeeds in 50 runs within the cutoff time. Therefore, PNFSat exhibits the best performance among these state-of-the-art solvers on the huge random 7-SAT instances.

4) ***Results on the 7-SAT_huge benchmark***: As reported in Table 4, the results show PNFSat dramatically outperforms its competitors. ProbSAT, WalkSATlm, YalSAT and FrwCBlm fail in all runs, while PNFSat succeeds in 65 runs, which indicates the scalability of the PNFSat algorithm.

5) ***Results on the 7-SAT_medium _SC benchmark***: Table 5 presents the experimental results of PNFSat and its competitors on medium random 7-SAT instances at phase transition. Although PNFSat solves a few less instances than other competitors, PNFSat has similar performance with its competitors on such instances.

6) ***Summarization for random 7-SAT***: Tables 1, 2, 3, 4 and 5 present the results of comparing PNFSat with ProbSAT, WalkSATlm, YalSAT, FrwCBlm as well as Score$_2$SAT on random 7-SAT instances from SAT Competition 2017, 2018 and 2019. PNFSat shows a substantial improvement over ProbSAT on these random 7-SAT instances. On all instance classes expect for the 7-SAT_ medium _SC, PNFSat achieves a higher success rate than ProbSAT does. Particularly, on the huge sized instances with 65≤*r*≤69, PNFSat succeeds in 40 runs while ProbSAT only succeeds in 20 runs. Moreover, PNFSat also significantly outperforms its competitions in terms of both success rate and run time on huge random 7-SAT instances, which indicates a substantial performance improvement of PNFSat over its competitions on these huge random 7-SAT instances. However, PNFSat does not show any notable improvement for ProbSAT on medium random 7-SAT instances (Table 5). On the other hand, PNFSat cannot rival state-of-the-art SLS solvers, such as the winners of SAT Competition 2017, on medium random 7-SAT instances at the phase transition, and thus further improved version is introduced in the subsequent section.

5.5 Further analysis of PNFSat and its alternative version PNFSat_alt

It might seem that TBF level is a relatively minor concern. In effect, however, it has an essential impact on the PNFSat algorithm. This is because when the algorithm based on the PNF strategy selects a variable to be flipped, there is sometimes more than one such selected variable, which is the same in two adjacent steps. Thus, in order to demonstrate the effectiveness of TBF in the PNF-TBF heuristic, we conducted experiments to compare PNFSat with an alternative version, called PNFSat_alt, as detailed below:

PNFSat_alt: this alternative version of PNFSat does not utilize the TBF component. In another word, this alternative version does not break ties of variables during the search process (i.e., removing lines 10-16 in Algorithm 2).

The parameter settings of PNFSat_alt is the same as that of PNFSat in the following experiments.

Empirical results for PNFSat and PNFSat_alt on all random 7-SAT instances from SAT Competition 2016, 2017 and 2018 are reported in Table 6. Each solver is performed for ten runs for each instance, as the instances in each ratio are enough to test the performance of the solvers [34]. The cutoff time for all runs is set to 5000 seconds.

Table 6: Experimental results of PNFSat and its alternative version on the random 7-SAT

Benchmark	#inst.	PNFSat_alt		PNFSat	
		#suc	par 10	#suc	par 10
7-SAT_huge_SC18	20	100	25012	**120**	**20028**
7-SAT_huge_SC17	20	100	25015	**120**	**20112**
7-SAT_huge_SC16	20	100	25013	**110**	**20516**
7-SAT_medium_SC	90	**470**	**24364**	420	25715

As is clear from Table 6, the performance of PNFSat_alt is much worse than PNFSat on huge random 7-SAT instances. Due to the TBF component, PNFSat gains a significant improvement over PNFSat_alt on huge random 7-SAT instances, while the performance of PNFSat_alt is better than PNFSat on medium random 7-SAT instances, which suggests that the new tie-breaking mechanism is likely suitable for solving huge random 7-SAT instances and not suitable for medium random 7-SAT instances at phase transition.

In order to demonstrate the effectiveness of PNFSat _alt, we conducted experiments to compare PNFSat_alt with other FRW solvers based on PD strategy including PNFSat, ProbSAT and YalSAT on the following medium random 7-SAT instances at the phase transition.

7-SAT_medium_Random: 7-SAT instances generated randomly according to the random k-SAT model (r=87.79, n=160, 170, 180, 60 instances, 20 for each size).

Each solver is performed for ten runs for each instance, and the cutoff time for all runs is set to 5000 seconds.

Empirical results for PNFSat_alt and other FRW solvers based on PD strategy on the 7-SAT_medium_random benchmark are reported in Table 7. As can be seen from Table 7, PNFSat_alt performs generally better than other PD strategy based FRW solvers, which indicates the effectiveness of PNFSat_alt. Particularly, on the medium random 7-SAT instances with n=160 and n=180, PNFSat_alt performs much better than those FRW solvers in terms of metrics, which confirms that our proposed PNF contributes to the performance of PNFSat_alt on the medium random 7-SAT instances, and TBF component is likely not suitable for medium random 7-SAT instances at phase transition, and indicates the scalability of the PNFSat_alt.

Table 7: Comparative results of PNFSat_alt and other PD strategy based FRW solvers on the 7-SAT_medium_random

Instance	ProbSAT		Yalsat		PNFSat		PNFSat_alt	
Class	#suc	par 10	#suc	par 10	#suc	par10	#suc	par 10
n=160	40	40318	40	40209	30	42718	**50**	**35888**
n =170	20	**45108**	0	n/a	0	n/a	20	45127
n =180	10	45507	10	45260	10	45525	**20**	**45149**

6 Improving PNFSat for random 5-SAT

The above section shows the excellent performance of PNFSat on huge random 7-SAT near the phase transition, and its variation PNFSat_alt on medium random 7-SAT at the phase transition. However, the performance of PNFSat and PNFSat_alt degrades on random 5-SAT instances (seen from Tables 8-12).

This section discusses the improvement of PNFSat for random 5-SAT instances with various sizes and ratios. To this end, we propose two variable selection heuristics called PN-PoF and Po-PNF respectively, which combine the PNF with the PoF utilized in the PNFSat algorithm in different ways, then they both are utilized to improve PNFSat, resulting in a new FRW algorithm called PN&PoFSat for SAT, along with detailed empirical evaluations of PN&PoFSat on a broad range of random 5-SAT instances.

6.1 PN-PoF heuristic and Po-PNF heuristic

The literature [16] has showed that the exponential delay in probability with growing *break* value might be too strong in the case of 3-SAT, thus, the PD strategy selects a variable x to be flipped according to the PoF of *break* value in the case of 3-SAT, and picks a variable x to be flipped according to the exponential function of *break* value for k-SAT with k>3.

There have been some limitations in previous PoF based FRW algorithms, which use PoF to handle the exponential decay for random k-SAT instances, but lose their capability and generality, especially for solving random k-SAT instances with k>3. Therefore, it is inadvisable to utilize either the exponential function or the PoF for random 5-SAT instances, which might have the similar performance with ProbSAT and polypower1.0 so that the improvement of PNFSat has no effect. Thus, an important issue in the PD based FRW algorithms is to seek a balance solution between the exponential function and the PoF.

Based on the above discussions, in order to improve PNFSat for random 5-SAT instances with various ratios, we propose two new probability functions, named PN-PoF and Po-PNF respectively, which reflect the different combination of the

PNF described in Section 4.1 and the PoF described in Section 2.2 as defined below. The further lead to two new variable selection heuristics respectively.

***Definition* 3**: For a CNF formula F, a complete assignment α to *var*(F), the pseudo normal-polynomial function of a variable x, denoted by *PN-PoF*, is a function on *var*(F) such that

$$PN-PoF(x,\alpha)=\begin{cases}PNF(x,a), & break(x,a)<d_1\\ PoF(x,a), & break(x,a)\geq d_1\end{cases} \quad (5)$$

where d_1 is a positive integer parameter.

***Definition* 4**: For a CNF formula F, a complete assignment α to *var*(F), the polynomial-pseudo normal function of a variable x, denoted by *Po-PNF*, is a function on *var*(F) such that

$$Po-PNF(x,\alpha)=\begin{cases}PoF(x,a), & break(x,a)<d_2\\ PNF(x,a), & break(x,a)\geq d_2\end{cases} \quad (6)$$

where d_2 is a positive integer parameter.

Note that since the distribution of the *break* value and the utilization rate of *break* value tends to be a Boltzmann function for solving k-SAT instances in WalkSATlm algorithm (described in Section 3), we utilize the Boltzmann function as a sample to guide the solution of the PD strategy based algorithms. Thus, the parameter d_1 or d_2 is determined based on the Boltzmann function.

Essentially PN-PoF and Po-PNF switch between two function, i.e., the PNF and the PoF, depending on the *break* value.

The PN-PoF heuristic or the Po-PNF heuristic prefers to select the variable to be flipped by the PN-PoF based PD strategy or the PN-PoF based PD strategy respectively. Flipping a variable by either of them minimizes the number of clauses from satisfiable to unsatisfied as soon as possible and handles the exponential decay. These two heuristics are described below:

In the PN-PoF heuristic:

-If the *break* value is less than the parameter d_1, the PD strategy is based on the PNF;
-otherwise, the PD strategy is based on the PoF.

In the Po-PNF heuristic:

- If the *break* value is less than the parameter d_2, the PD strategy is based on the PoF;
- otherwise, the PD strategy is based on the PNF.

Since the algorithm calls PN-PoF heuristic and Po-PNF heuristic according to clause-to-variable ratio of SAT respectively, the parameter settings for d_1 and d_2 may be different from each other in two heuristics.

6.2 PN&PoFSat algorithm

In this section, we modify the PD strategy of PNFSat by combined use of the PN-PoF and the Po-PNF and obtain a new algorithm which refers to as PN&PoFSat. The pseudo-codes of PN&PoFSat is given in Algorithm 3.

PN&PoFSat differs from PNFSat in the following aspect: although both algorithms utilize the PD strategy, PN&PoFSat uses both the PN-PoF heuristic (lines 8 and 9 in Algorithm 3) and the Po-PNF heuristic (lines 10 and 11 in Algorithm 3), to use which is dependent on the preset conjectured threshold ratio of clause-to-variable for solving random 5-SAT instances, while PNFSat only use the PNF based PD strategy.

Algorithm 3: PN&PoFSat (F)

```
Input: CNF-formula F, MaxTries, MaxSteps
Output: A satisfying assignment α of F, or Unknown
1  for i = 1 to MaxTries do
2      α ←a generated assignment for F by variable allocation value function;
3      for j = 1 to MaxSteps do
4          if α satisfies F then
5              return α;
6          C ←an unsatisfied clause selected randomly;
7          for x in C do
8              if r equal to the conjectured threshold ratio of the solubility
                   phase transition then
9                  f(x, α )←compute PN-PoF (x, α);
10             else
11                 f(x, α)←compute Po-PNF(x, α);
12         v ←random variable x according to probability f(x,α)/Σ_{z∈C} f(z,α);
13         if  v= =bestVar then
14             if numFalse<R then
15                 bestVar ←random variable x in TBFVar(c);
16             else
17                 bestVar ←random variable x in TBFVar(α);
18         else
19             bestVar=v;
20         flip(bestVar);
21 return Unknown;
```

6.3 Evaluations of PN&PoFSat on random 5-SAT instances

In this subsection, we carry out extensive experiments to evaluate PN&PoFSat on random 5-SAT instances at and near phase transition. First, we compare PN&PoFSat with PNFSat as well as state-of-the-art SLS solvers on random 5-SAT instances at and near the phase transition from SAT Competitions in 2016, 2017 and 2018. Then, we compare PN&PoFSat with state-of-the-art SLS solvers on large-sized threshold and 5-SAT_huge random instances generated randomly at and near the threshold of phase transition.

6.3.1 Benchmark and experimental preliminaries

In the experiments in this section, all benchmark instances are generated according to the random k-SAT model at and near the threshold ratio of the solubility phase transition. Specifically, we adopt the following seven benchmarks.

1) **5-SAT_huge_SC18**: The benchmark includes all 20 huge random 5-SAT instances with $16.0 \leq r \leq 16.8$, $17.0 \leq r \leq 17.8$, $18.0 \leq r \leq 18.8$ and $19.0 \leq r \leq 19.8$ (n=250000, five instances each class) from SAT Competition 2018.

2) **5-SAT_huge_SC17**: The benchmark contains all huge 5-SAT instances with $16.0 \leq r \leq 16.8$, $17.0 \leq r \leq 17.8$, 18.0 ≤18.8 and $19.0 \leq r \leq 19.8$ (n=250000, five instances each class) from SAT Competition 2017.

3) **5-SAT_huge _SC16:** The benchmark consists of all 20

huge random 5-SAT instances with various ratio ($55 \leq r \leq 74$, n=250000, one instance each ratio) from SAT Competition 2016.

4) **5-SAT_medium_SC18**: The benchmark contains medium random 5-SAT instances (r=87.79, n=250, ten instances each size) from SAT Competition 2018.

5) **5-SAT_medium_ SC**: The benchmark consists of all 80 random 5-SAT instances with $200 \leq n \leq 290$, 300≤n≤390, $400 \leq n \leq 490$and 500≤n≤590 (r=21.117, 20 instances each class) from SAT Competitions in 2016 and 2017.

6) **Large-sized threshold**: Random 5-SAT instances at the threshold ratio of phase transition, generated random by the random k-SAT generator[8] utilized in SAT Competitions 2016, 2017 and 2018 (r=21.117, n=550, 600, 650, 120 instances, 40 for each size). These instances are divided into two categories: the training set and test set, both of which have 20 instances for each 5-SAT class.

7) **5-SAT_huge**: Random 5-SAT instances near the threshold ratio of phase transition, generated random by the random k-SAT generator (r=18.2, 18.4, 18.6, n=250000, 120 instances, 40 for each size). These instances are divided into two categories: the training set and test set, both of which have 20 instances for each 5-SAT class.

Note that the training set is only utilized to tune the parameters in PN&PoFSat, and then PN&PoFSat with the tuned parameters is evaluated on random 5-SAT instances at and near the threshold ratio of phase transition from SAT Competitions 2016, 2017 and 2018 and the test set in large-sized threshold benchmark and 5-SAT_huge benchmark.

The PN&PoFSat algorithm is developed on the top of PNFSat, and thus is implemented in C/C++. For the parameters of Vav function in PN&PoFSat and PNFSat, we use the default parameter setting tuned on random 5-SAT instances in the literature [20]. For the three parameters R, ε and c_b in PN&PoFSat, we set R to 3, ε to 1 and c_b to 3.7 as constants. The parameter d_1 and d_2 for the PN-PoF heuristic and the Po-PNF heuristic respectively in PN&PoFSat are tuned based on all random 5-SAT instances from SAT Competitions 2016, 2107 and 2018 as well as the training set of the large-sized threshold benchmark and 5-SAT_huge benchmark.

First, we observe the Boltzmann function in Fig. 3 to find preferred parameter for huge random 5-SAT instances near the threshold ratio of phase transition. The obvious monotonic decreasing interval of the Boltzmann function is between 0 and 4, and when the *break* value is greater than 3, the trend of the function is very flat, i.e., the difference in Boltzmann function under different *break* values is small. Hence, the turning point of the Boltzmann function trend is at the point where the *break* value is equal to 3. Thus, we test d_1 (also d_2)=2, 3, 4 to find the most efficient one for huge random 5-SAT instances, and test the PN-PoF heuristic and the Po-PNF heuristic respectively in PN&PoFSat to find the most efficient one for different ratios. The preliminary results show that d_1=4 for the PN-PoF heuristic is the best for huge random 5-SAT with r<18; d_2=3 for the Po-PNF heuristic is the best for huge random 5-SAT with $18.0 \leq r < 21.0$.

Then we observe the Boltzmann function in Fig. 4 to find preferred parameter for medium random 5-SAT instances at the threshold ratio of phase transition. When the *break* value is greater than 4, the difference in Boltzmann function under different break values is small. The turning point of the Boltzmann function trend is at the point where the *break* value is equal to 4. Hence, we test d_1 =2, 3, 4, 5 to find the most efficient one for medium random 5-SAT instances, and test the PN-PoF and the Po-PNF heuristic respectively in PN&PoFSat to find the most efficient one for different sizes.

The preliminary results show that the d_1=4 for the PN-PoF heuristic is the best for medium random 5-SAT at the threshold ratio of phase transition with n<330; d_1=2 for the PN-PoF heuristic is the best for medium random 5-SAT at the threshold ratio of phase transition with$330 \leq n < 430$; d_1=5 for the PN-PoF heuristic is the best for medium random 5-SAT at the threshold ratio of phase transition with $n \geq 430$.

For the 5-SAT_huge_SC18 benchmark, 5-SAT_huge_SC17 benchmark, 5-SAT_ huge_SC16 benchmark and 5-SAT_ medium_SC18 benchmark, each solver is performed for ten runs for each instance. For the 5-SAT_medium_SC benchmark and large-sized threshold benchmark as well as 5-SAT_huge benchmark, each solver is performed for five runs for each instance. For large-sized threshold benchmark and 5-SAT_huge benchmark, the cutoff time for all runs is set to 2000 s, and for other benchmarks, the cutoff time for all runs is set to 5000 s (as in SAT Competition 2016, 2017 and 2018).

6.3.2 Experimental results

The experiments results are summarized below:

1) ***Results on the* 5-*SAT_huge_SC*18 *benchmark***: Table 8 shows experimental results on the 5-SAT_huge benchmark. As is clear from Table 8, PN&PoFSat shows significantly better performance than other competitors on the whole instances interms of both successful runs and average run time. Particularly, on the random 5-SAT instances with 18.0≤r≤18.8 instances class, which are of the largest size in SAT competitions on random 5-SAT instances, the runtime of PN&PoFSat Sat is about 1.5 times less than of YalSAT, and about 2 orders of magnitudes less than those of other state-of-the-art SLS solvers. Also, for the whole benchmark, PN&PoFSat solves 130 runs, compared to 120 for YalSAT, 110 for WalkSATlm, FrwCBlm and ProbSAT respectively.

2) ***Results on the* 5-*SAT_huge_SC*17 *benchmark***: To show the robustness of PN&PoFSat, we compare PN&PoFSat with its competitors on the huge random 5-SAT benchmark from SAT Competition 2017. Table 9 presents the results of PN&PoFSat and its competitors on this benchmark.

According to the results, PN&PoFSat shows significant superiority over PNFSat and significantly performs better than its competitors, which confirms the robustness of PN&PoFSat on huge random 5-SAT instances.

[8]https://sourceforge.net/projects/ksatgenerator/

Table 8: Comparative results of PN&PoFSat and its competitors on the 5-SAT_huge_SC18

Instance Class	WalkSATlm		FrwCBlm		ProbSAT		Yalsat		PNFSat		PN&PoFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
16.0≤*r*≤16.8	50	25	50	35	50	18	50	44	50	18	**50**	**15**
17.0≤*r*≤17.8	50	295	50	376	50	63	50	106	50	111	**50**	**62**
18.0≤*r*≤18.8	10	40329	10	40057	10	40033	20	30156	0	n/a	**30**	**20088**
19.0≤*r*≤19.8	0	n/a	0	n/a	0	n/a	0	n/a	0	n/a	**0**	**n/a**
Overall	110	22662	100	25211	110	22524	120	20077	100	25032	**130**	**17542**

Table 9: Comparative results of PN&PoFSat and its competitors on the 5-SAT_huge_SC17

Instance Class	WalkSATlm		FrwCBlm		ProbSAT		Yalsat		PNFSat		PN&PoFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
16.0≤*r*≤16.8	50	34	50	40	50	18	50	57	50	17	**50**	**15**
17.0≤*r*≤17.8	40	10378	50	416	50	**43**	50	137	50	258	**50**	77
18.0≤*r*≤18.8	0	n/a	10	42119	10	40210	20	30394	0	n/a	**30**	**20083**
19.0≤*r*≤19.8	0	n/a	0	n/a	0	n/a	0	n/a	0	n/a	**n/a**	**n/a**
Overall	90	27603	110	26900	110	22526	120	20147	100	25138	**130**	**17544**

Table 10: Comparative results of PN&PoFSat and its competitors on the 5-SAT_huge_SC16

Instance Class	WalkSATlm		FrwCBlm		ProbSAT		Yalsat		PNFSat		PN&PoFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
16.0≤*r*≤16.8	50	23	50	42	50	18	50	43	50	18	**50**	**15**
17.0≤*r*≤17.8	50	306	50	467	50	**46**	50	107	50	117	**50**	68
18.0≤*r*≤18.8	20	31320	10	40617	10	40062	10	40050	0	n/a	**40**	**10396**
19.0≤*r*≤19.8	0	n/a	0	n/a	0	n/a	0	n/a	0	n/a	**0**	**n/a**
Overall	120	20412	110	22782	110	22531	110	22550	100	25034	**140**	**15120**

Table 11: Comparative results of PN&PoFSat and its competitors on the 5-SAT_medium_SC18

#Total runs	WalkSATlm		FrwCBlm		ProbSAT		Yalsat		Score$_2$SAT		PNFSat		PN&PoFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
100	70	15023	70	15070	80	10409	80	10222	70	15006	70	15010	**90**	**5148**

Table 12: Comparative results of PN&PoFSat and its competitors on the 5-SAT_medium_SC

Instance Class	WalkSATlm		ProbSAT		Yalsat		Score$_2$SAT		PNFSat		PN&PoFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
200≤*n*≤290	60	20290	45	27503	55	22526	60	20050	45	27517	**60**	**20021**
300≤*n*≤390	35	32535	40	30014	40	30051	**45**	**27624**	40	30015	40	30014
400≤*n*≤490	15	42792	20	40163	20	40248	25	37782	20	40041	**25**	**37665**
500≤*n*≤590	15	37599	25	37795	20	40251	20	40242	20	40295	**30**	**35302**
Overall	125	34554	130	33868	135	33269	150	31424	125	34467	**155**	**30750**

Table 13: Comparative results of PN&PoFSat and its competitors on the large-sized threshold

Instance Class	WalkSATlm		ProbSAT		Yalsat		Score$_2$SAT		PN&PoFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
n =550	5	47598	10	45051	5	47581	10	45079	**20**	**40278**
n =600	10	45139	15	**42587**	10	45124	10	45124	15	42665
n =650	5	47571	5	47534	5	47568	25	37782	**10**	**45127**

Table 14: Comparative results of PN&PoFSat and its competitors on the 5-SAT_huge

Instance Class	WalkSATlm		ProbSAT		Yalsat		FrwCBlm		PN&PoFSat	
	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10	#suc	par 10
r=18.2	35	34089	0	n/a	0	n/a	5	47738	**100**	**134**
r=18.4	0	n/a	0	n/a	0	n/a	0	n/a	**100**	**255**
r=18.6	0	n/a	0	n/a	0	n/a	0	n/a	**1**	**47661**

3) ***Results on the 5-SAT_huge_SC16 benchmark:*** To solve the huge random 5-SAT instances, Table 10 presents the results of PN&PoFSat and its competitors on this benchmark. In terms of success runs, PN&PoFSat stands out the best solver. The only instance class for which PNFSat does not give the best performance is random 5-SAT instances with 17.0≤*r*≤17.8 in terms of average run time. Nevertheless, on this instance class, PN&PoFSat has similar performance as the best solver ProbSAT, spending 22 more time. For the whole benchmark, PN&PoFSat solves 30 runs more than FrwCBlm, ProbSAT and

YalSAT do respectively, 20 runs more than WalkSATlm does. Therefore, PN&PoFSat exhibits the best performance among these state-of-the-art solvers on the huge random 5-SAT instances.

4) ***Results on the* 5-*SAT_medium_SC*18 *benchmark*:** The comparative results on the 5-SAT_medium_SC18 benchmark are presented in Table 11. PN&PoFSat significantly outperforms other solvers on all these medium random 5-SAT instances of SAT Competition 2018 in terms of metrics. Specially, PN&PoFSat achieves success runs of 90, which are 20 runs more than those of WalkSATlm, FrwCBlm, $Score_2SAT$ and PNFSat respectively, and 10 runs more than those of ProbSAT and YalSAT respectively.

5) ***Results on the* 5-*SAT_medium_SC benchmark***: Table12 shows experimental results on the 5-SAT_medium_SC benchmark. PN&PoFSat solves a few more instances than its competitors. Overall, PN&PoFSat succeeds in 155 runs, compared to 125 for both WalkSATllm and PNFSat, 130 for ProbSAT, 135 for YalSAT and 150 for $Score_2SAT$. Further, observation shows that, PN&PoFSat has similar performance with $Score_2SAT$ on random 5-SAT instances with $300 \le n \le 390$ instance class.

6) ***Results on the large-sized threshold benchmark***: To measure the performance of PN&PoFSat on random phase-transition 5-SAT instances more accurately, we additionally test PN&PoFSat on the test set of the Large-sized Threshold benchmark, compared with WalkSATlm, FrwCBlm, YalSAT $Score_2SAT$ and ProbSAT. The results are presented in Table 13. For random 5-SAT instances with n=600, although PN&PoFSat has more run time, PN&PoFSat and ProbSAT solve the same number of instances. For the remaining instance class, PN&PoFSat solves the most instances. Particularly, PN&PoFSat shows significantly superior performance than other solves on random 5-SAT instances with n=550, where it succeeds in 20 runs, while ProbSAT and $Score_2SAT$ both succeed in 10 runs and YalSAT and FrwCBlm both succeed in 5 runs, which indicates the scalability of the PN&PoFSat algorithm.

7) ***Results on the* 5-*SAT_huge benchmark***: The experimental results on the 5-SAT_Huge benchmark are presented in Table 14. It is encouraging to see the performance of PN&PoFSat remains surprisingly good on these very huge 5-SAT instances, where state-of-the-art solvers show very poor performance. PN&PoFSat solves these 5-SAT instances with up to 18.4 ratio consistently (i.e., with 100% success rate). Furthermore, PN&PoFSat succeeds in one run for the huge 5-SAT instances with r=18.6 respectively, whereas all its competitors fail to find a solution for any of these instances, which indicates the scalability of the PN&PoFSat algorithm.

In summary, the experimental results show PN&PoFSat consistently outperforms WalkSATlm, FrwCBlm, ProbSAT, YalSAT, $Score_2SAT$ and PNFSat on solving random 5-SAT instances with various ratios and sizes. We believe that the better performance of PN&PoFSat is mainly attributed to the combination of both the PN-PoF heuristic and the Po-PNF heuristic.

6.4 Experimental analysis on PN&PoFSat

We conduct further empirical analyses to study effectiveness of the PN-PoF heuristic and the Po-PNF heuristic in PN&PoFSat. We compare PN&PoFSat with its two alternatives on all random 5-SAT instances from SAT Competitions in 2016, 2017 and 2018. Two alternative solvers are described below:

(i) $PN\&PoFSat_1$: this alternative version of PN&PoFSat does not utilize the PN-PoF heuristic. In another word, this alternative version only uses the Po-PNF heuristic during the search process (i.e., replacing PN-PoF heuristic with Po-PNF heuristic i.e., lines 10-11 in Algorithm 3).

(ii) $PN\&PoFSat_2$: this alternative version of PN&PoFSat does not utilize the Po-PNF heuristic. In another word, this alternative version only uses the PN-PoF heuristic during the search process (i.e., replacing Po-PNF heuristic with PN-PoF heuristic i.e., lines 8-9 in Algorithm 3).

We use of the default value of PN&PoFSat as the parameter settings of $PN\&PoFSat_1$ and $PN\&PoFSat_2$, but they differ on the variable selection heuristic based on PD. $PN\&PoFSat_1$ only uses the Po-PNF heuristic, while $PN\&PoFSat_2$ only employs the PN-PoF heuristic. Thus, the comparison between $PN\&PoFSat_1$ and $PN\&PoFSat_2$ is interesting, as it can show the contribution of the two heuristics propose in PN&PoFSat.

Empirical results for $PN\&PoFSat_1$, $PN\&PoFSat_2$ and PN&PoFSat on all random 5-SAT instances from SAT Competitions in 2016, 2017 and 2018 are reported in Table 15. Each solver is performed for ten runs on each instance. The cutoff time for all runs is set to 5000 seconds.

As is clear from Table 15, the performance of PN&PoFSat is much better than $PN\&PoFSat_1$ and $PN\&PoFSat_2$ on random 5-SAT instances. The comparison between $PN\&PoFSat_1$ and $PN\&PoFSat_2$ illustrates that Po-PNF heuristic and PN-PoF heuristic have their superiority in different situations. For the medium random 5-SAT instances at the phase transition, $PN\&PoFSat_1$ performs worse than $PN\&PoFSat_2$. For the huge random 5-SAT near the phase transition, $PN\&PoFSat_1$ outperforms $PN\&PoFSat_2$. Based on this observation, we conjecture that for local search algorithms, if the ratio of clause-to-variable is equal to the conjectured threshold ratio of the solubility phase transition, it is better to utilize the PN-PoF heuristic than the Po-PNF heuristic, and otherwise the Po-PNF heuristic is of more benefit.

7 An integrated the PDSat solver and results on random k-SAT instances with long clauses

As PNFSat, PNFSat _alt and PN&PoFSat are all PD st based algorithms, we combine them and obtain a flexible local search SAT solver called PDSat to handle different k-SAT instances. Specifically, PDSat adopts PN&PoFSat to solve random 5-SAT, and PNFSat _alt to solve medium random 7-SAT instances with r equal to the conjectured threshold ratio of the solubility phase transition, and PNFSat to solve huge random 7-SAT instances near the phase transition.

Table 15: Experimental results of PN&PoFSat and its alternative version on the random 5-SAT

Benchmark	#inst.	PN&PoFSat		$PN\&PoFSat_1$		$PN\&PoFSat_2$	
		#suc	par 10	#suc	par 10	#suc	par 10
5SAT_huge_SC18	20	**130**	**17545**	**130**	**17543**	100	50042
5SAT_huge_SC17	20	**130**	**17545**	**130**	**17546**	100	50049
5SAT_huge_SC16	20	**140**	**15123**	**140**	**15121**	100	50043
5SAT_medium_SC18	10	**90**	**5149**	70	15061	**90**	**5150**
5SAT_medium_SC	80	**310**	**30753**	200	37615	**310**	**30752**

The PDSat is implemented in C/C++. The parameters for PNFSat, PNFSat _alt and PN&PoFSat are set as the same as those in the experiments in Sections 5 and 6 respectively.

In this section, we present the experimental results of PDSat on random *k*-SAT instances with long clauses from the random track of SAT Competitions in 2016, 2017 and 2018.

7.1 Experiment preliminaries of PDSat for random k-SAT with long clauses benchmarks

SAT Competition is a competitive event for solvers of the SAT problem. They have been held yearly 12 times starting from 2002. All random *k*-SAT with long clauses instances from the SAT Competitions in 2017, 2018 and 2019 are generated according to the random *k*-SAT generator at and near the threshold ratios. We adopt the following three benchmarks

1) ***k*-SAT_SC16**: all medium and huge random *k*-SAT instances with long clauses from SAT Competition 2016, and each *k*-SAT, the instances contains various sizes and ratios. The details of the benchmark are given in Table 16.

2) ***k*-SAT_SC17**: all 120 medium and huge random *k*-SAT instances (all 60 huge and medium 5-SAT instances described in Section 6, 60 huge and medium random 7-SAT instances described in Section 5, from SAT Competition 2017.

3) ***k*-SAT_SC18**: all random *k*-SAT instances with long clauses from SAT Competition 2018, and each *k*-SAT, the instances have various sizes and ratios. The details of the benchmark are given in Table 17

We compare PDSat with four state-of-the-art FRW solvers, i.e., WalkSATlm, FrwCBlm, YalSAT, ProbSAT and four two-mode SLS solvers, containing Sparrow [15], DCCASat [21], CSCCSat [11], $Score_2SAT$. Sparrow is the winner of random track of SAT Competition 2011, and Sparrow combined with a complete algorithm to form a new solver Sparrow2Riss winning the random track of SAT competition 2018, but we did not compare the top three solvers of the random track of SAT competition 2018 with PDSat. The top five solvers mainly utilize the complete algorithms, but ours are based on incomplete algorithms. These solvers did not solve any instances for the medium and huge instances of the SAT 2018[9]competition, except the champion solver Sparrow2Riss can solve a small number of these instances, so these solvers don't apply to the medium and huge random *k*-SAT instances. DCCASat is still highly competitive with state-of-the-art solvers on random *k*-SAT instances. CSCCSat won the bronze medal and silver medal in the random track of SAT Competitions in 2014 and 2016 respectively, but its performance outperforms the winner of SAT Competition 2017 on medium and huge random *k*-SAT instances with various ratios. For the *k*-SAT_SC16 benchmark and *k*-SAT_SC17, each solver is performed for five runs for each instance. For the *k*-SAT_SC18, each solver is performed for ten runs for each instance. The cutoff time for all runs is set to 5000 s.

[9]http://sat2018.forsyte.tuwien.ac.at/index.php?cat=results

7.2 Experimental results of PDSat for random k-SAT with long clauses benchmarks

The experiments results are summarized below:

1) ***Results on the k-SAT_SC*16 *benchmark*:** First, each solver is performed for one run on each instance. We present the CPU time distributions for PDSat and its competitors in Fig. 5 below. As can be seen from Fig. 5, PDSat outperforms its competitors. Then to show the robustness of PDSat on random *k*-SAT instances with long clauses and various sizes and ratios, each solver is performed for five runs on each instance, and the results are reported in Table 18. Seen from Table 18, PDSat stands out as the best solver and significantly performs better than its all FRW and two-mode competitors in terms of both metrics. Overall, PDSat succeeds in 285 runs, and PDSat solves the most instances, which illustrates its robustness.

2) ***Results on the k-SAT_SC*17 *benchmark*:** Table 19 reports the number of successful runs and PAR 10 for PDSat and its competitors on the *k*-SAT_SC17 benchmark. The results show PDSat significantly outperforms its competitors in terms of both metrics. On the whole benchmarks, PDSat succeeds in 315 runs. Further observation in Fig. 6 below shows that PDSat takes less than about 2000 seconds than other solvers do. More encouragingly, PDSat solves 75 runs more Sparrow does, 60 runs more than YalSAT does, 50 runs more than both FrwCBlm and WalkSATlm do, 45 runs more than DCCASat and ProbSAT does respectively, 30 runs more than both $Score_2SAT$ and CSCCSat do.

3) ***Results on the k-SAT_SC*18 *benchmark*:** To evaluate the genuine solving ability on medium and huge (at and near the phase transition) random *k*-SAT instances with long clauses, we compare PDSat with its competitors on the *k*-SAT_SC18 benchmark, and the results are reported in Table 20. PDSat stands out as the best solver and significantly outperforms its all FRW and two-mode competitors in terms of both metrics on this benchmark. PDSat succeeds in 450 (out of 900) runs, 70 more than the second solver namely ProbSAT does, which indicates its robustness on medium and huge random *k*-SAT instances with long clauses.

Table 16 The instances numbers, ratio and sizes for each *k*-SAT with long clauses in the k-SAT_SC16 benchmark

	5-SAT		7-SAT	
	medium	huge	medium	huge
#inst.	40	20	40	20
ratio	21.117	$r\in \{16.0, 16.2, \dots, 19.8\}$	87.79	$r\in \{55.0, 56.0, \dots, 74.0\}$
size	$n \in \{200, 210, \dots, 590\}$	250000	$n \in \{90, 92, \dots, 168\}$	50000

Table 17: The instances numbers, ratio and sizes for each *k*-SAT with long clauses in the k-SAT_SC18 benchmark

	5-SAT		7-SAT	
	medium	huge	medium	huge
#inst.	10	20	10	20
ratio	21.117	$r\in \{16.0, 16.2, \dots, 19.8\}$	87.79	$r\in \{55.0, 56.0, \dots, 74.0\}$
size	250	250000	120	50000

Table 18: Comparative results of PDSat and its competitors on the *k*-SAT_SC16 benchmark

Instance Class	DCCASat #suc par 10	Sparrow #suc par 10	FrwCBlm #suc par 10	YalSAT #suc par 10	CSCCSat #suc par 10	ProbSAT #suc par 10	WalkSATlm #suc par 10	$Score_2SAT$ #suc par 10	PDSat #suc par 10
#Total	240 30187	200 33480	240 30191	255 28907	275 27325	270 27651	275 27430	275 27337	**285** **26430**

Table 19: Comparative results of PDSat and its competitors on the *k*-SAT_SC17 benchmark

Instance Class	DCCASat #suc par 10	Sparrow #suc par 10	FrwCBlm #suc par 10	YalSAT #suc par 10	WalkSATlm #suc par 10	ProbSAT #suc par 10	$Score_2SAT$ #suc par 10	CSCCSat #suc par 10	PDSat #suc par 10
#Total	270 27773	240 30328	265 28220	255 28876	265 28235	270 27633	285 26502	285 26516	**315** **24007**

Table 20: Comparative results of PDSat and its competitors on the *k*-SAT_SC18 benchmark

Instance Class	DCCASat #suc par 10	Sparrow #suc par 10	YalSAT #suc par 10	FrwCBlm #suc par 10	WalkSATlm #suc par 10	$Score_2SAT$ #suc par 10	CSCCSat #suc par 10	ProbSAT #suc par 10	PDSat #suc par 10
#Total	350 21175	290 26041	360 20162	350 21027	360 20200	340 10902	360 20205	380 18581	**450** **12676**

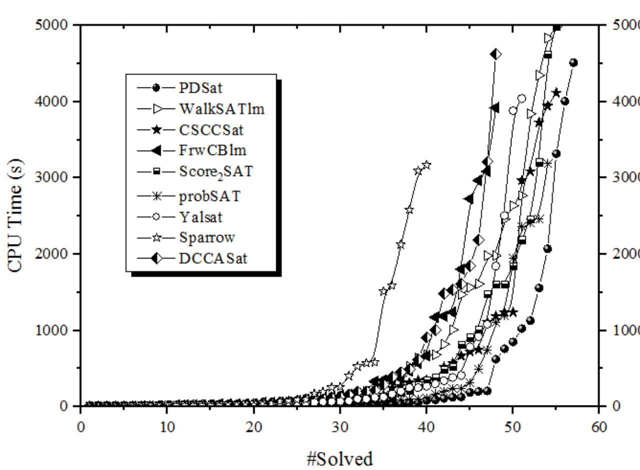

Fig. 5 Comparison of run time distributions on the SAT Competition 2016 benchmark consisting of all medium and huge random *k*-SAT instances with long clauses with a cutoff time of 5000 seconds.

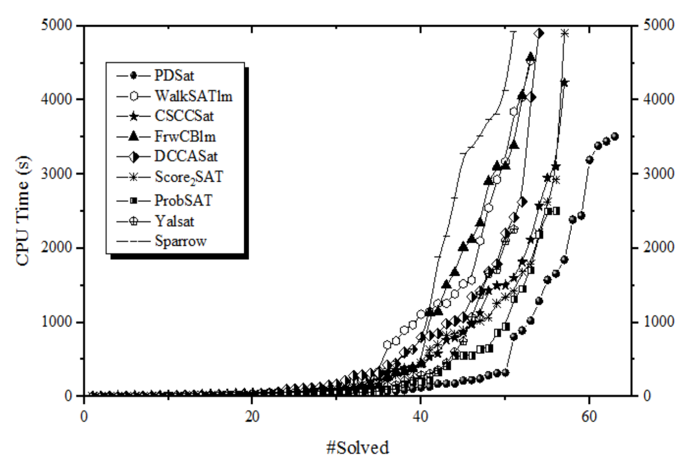

Fig. 6 Comparison of run time distributions on the SAT Competition 2017 benchmark consisting of all medium and huge random *k*-SAT instances with long clauses with a cutoff time of 5000 seconds.

The good performance of PDSat on the SAT Competition 2018 benchmark is also clearly illustrated by Fig. 7, which summarizes the run time distributions of the solvers on this benchmark, and each solver is performed for one run for each instance. These promising results of PDSat confirm the effectiveness of PNF, Po-PNF, PN-PoF, variable allocation value function and the new tie- breaking mechanism.

In summary, the experiments show that PDSat consistently outperforms WalkSATlm, FrwCBlm, ProbSAT, YalSAT, $Score_2SAT$ CSCCSat, DCCASat and Sparrow on solving medium and huge random *k*-SAT instances with long clauses and various ratios and sizes in terms of both metrics, which confirms the robustness of PDSat on *k*-SAT instances with long clauses.

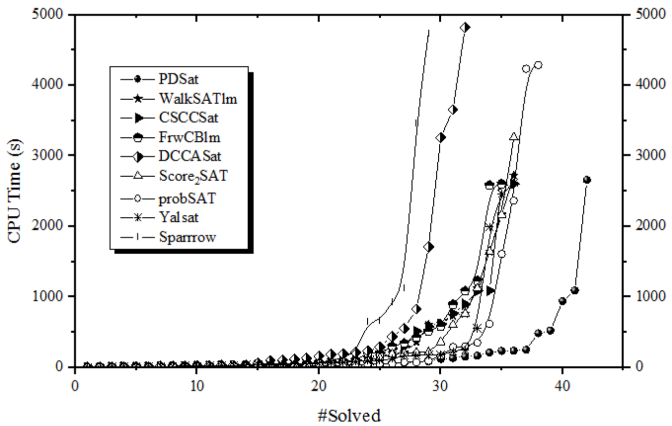

Fig. 7 Comparison of run time distributions on the SAT Competition 2018 benchmark consisting of all medium and huge random *k*-SAT instances with long clauses with a cutoff time of 5000 seconds.

8 Further discussions

Some further discussions are given below to clarify some issues and highlight some important cases.

8.1 Links between different algorithms and their capability and applicability

Fig. 8 below summarizes the strategies of each algorithm, their links among each other and their capability and applicability.

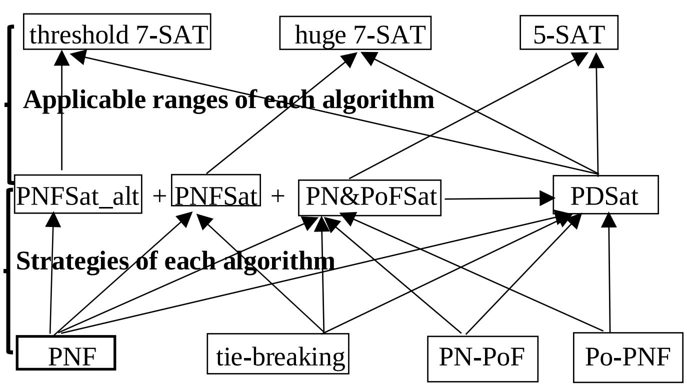

Fig. 8 The strategies of each algorithm and applicable ranges of each algorithm on PDSat.

Note that threshold 7-SAT benchmark includes the 7-SAT instances with *r* equal to the conjectured threshold ratio of the solubility phase transition (i.e., *r*=87.79), and huge 7-SAT benchmark contains the 7-SAT instances with *r*<87.79 and *n*=50000, and 5-SAT benchmark includes the 5-SAT instances with various ratios and variables. PN-PoF and Po-PNF is based on PNF and PoF.

According to the Fig. 8, the PDSat algorithm includes four implementation of strategies - PNF, the new tie-breaking, PN-PoF, and Po-PNF respectively. Next, we provide further discussion about each implementation of PDSat. Then we conduct further analysis to provide more insights into the PD strategy based on PNF, PN-PoF, Po-PNF, the Vav function and the TBF mechanism. Specifically, further experiments are conducted to reveal the relationships among the PD strategies based on PNF, Po-PNF and PN-PoF and the other two related heuristics. PD strategy based on the PoF itself is not suitable on random *k*-SAT instances with long clauses.

8.2 Approximate implementation of PD strategies based on PNF, Po-PNF, and PN-PoF

In this paper, the implementation of PNF described in Sections 5 and 6, and Po-PNF and PN-PoF described in Section 6 are also approximate strategies.

Inspired by the approximate implementation of PD strategy [16], and the fitting distribution of *break* value in WalkSATlm, which significantly decrease the time complexity of the accurate implementation of PD strategy, we firstly propose an accurate implementation of PD strategies based on PNF, Po-PNF and PN-PoF, which computes the probability of *break* value of all variables in an unsatisfied clause *c* selected under a complete assignment *α*. The maintenance of the accurate implementation is described as follows: whenever a variable *x* is flipped during the search, firstly *x*'s *break* value is stored. Then each clause $c \in C(x)$ is checked whether *c*'s state is changed (from unsatisfied to 1-satisfied, from 1-satisfied to 2-satisfied, from 2-satisfied to 1-satisfied, or from 1-satisfied to unsatisfied) by flipping a variable *y*. If it is the case (*c*'s state is changed by flipping the variable *y*), for each variable *x* in *c*, *x*'s *break* value is updated. Then if *x* appears in the subsequent unsatisfied clause selected; *x*'s probability would be updated.

We use $L(x)$ to denote the occurrence number of a variable *x*. As variable *x* appears in each clause for $C(x)$, thus $L(x)$ is equal to $|C(x)|$.

Note that the discussions below are based on the condition that *F* is a random *k*-SAT instance with *n* variables and *m* clauses ($r=m/n$). For each clause *c*, the number of all variables is equal to *k*, i.e., $E(|c|) = k$. For each variable $x \in var(F)$, $E(|L(x)|)$ is about equal to $k * m/n = k*r$, thus it is easy to derive that $E(|C(x)|)$ is also about equal to $k*r$.

For the accurate implementation of PD strategies based on PNF, Po-PNF and PN-PoF, the time complexity of computing the PNF, Po-PNF or Po-PNF is $O(E(|c|)) = O(k)$, and the time complexity of maintenance is $O(E(|c|))+ O(E(|C(x)|)= O(k)+ O(k*r)= O(k*r)$. All the time complexities of maintenance of the approximate implementation of PD strategy computed only by the *break* value are $O(k*r)$.

Since PNF is a potential idea for escaping the cycling problem and the FRW paradigm shows efficiency on selecting a flipping variable to be flipped in each search step, thus it is interesting to apply PNF to FRW. While the existing PoF strategy is ineffective to handle the exponential delay on random *k*-SAT instances with long clauses when applying to FRW, Po-PNF and PN-PoF show effectiveness when combining into FRW. The possible reason is that the fitting distribution of the *break* value in WalkSATlm helps FRW algorithms to combine PoF strategy with the PNF strategy and decrease the exponential delay, *Vav* function helps FRW algorithms to decrease blind initial assignment, and the TBF mechanism helps FRW algorithms to avoid the useless flips in adjacent steps and thus lead FRW algorithms to the promising search spaces.

8.3 Empirical analyzes on PoF, PNF, TBF mechanism and Vav function

To show the superiorities of PNF, Po-PNF and PN-PoF over PoF on FRW algorithms for random *k*-SAT instances with long clauses, and demonstrate the relationship among PD strategy, TBF mechanism, and Vav function in the PDSat, we directly replace PNF, Po-PNF, PN-PoF with PoF, resulting in an alternative version called PDSat_PoF; only utilize PNF instead of other Po-PNF, PN-PoF and PoF, resulting in an algorithm called PDSat_PNF; do not use the *Vav* function, resulting in an alternative version namely PDSat_nva; do not utilize the *TBF* mechanism, resulting in an algorithm called PDSat_nTBF; and do not use the *TBF* mechanism and *Vav* function, resulting in another version namely PDSat_nvt.

For PDSat_PoF on solving *k*-SAT instances with long clauses, we set the setting of c_b as the ones used in ProbSAT [16], and ε to 1 as the constant. The parameters of five alternative algorithms are set according to PDSat. The cutoff time for all runs is set to 5000 s. We run each solver ten times for each instance, as the instances in each ratio are enough to test the performance of the solvers [34].

Then, we compare PDSat with the five alternative versions on extensive medium and huge random *k*-SAT instances, including the *k*-SAT_SC16 benchmark, the *k*-SAT_SC17 benchmark and the *k*-SAT_SC18 benchmark described in Section 7. The experimental results are presented in Table 21.

Table 21: Comparative results of PDSat and its alternative versions on the *k*-SAT benchmark

Solver	#solved			par 10
	5-SAT	7-SAT	#Total Runs	
#inst.	150	150	3000	
PDSat	**800**	**820**	**1620**	**23206**
PDSAT _nva	760	730	1490	25397
PDSAT _nTBF	630	800	1430	26363
PDSAT _nvt	640	770	1410	26698
PDSAT _PoF	630	560	1190	30276
PDSAT _PNF	570	940	1510	25035

According to the experimental results, it is apparent that PNF exists the exponential decay with growing *break* value in case of random 3-SAT. Although PoF might handle the exponential decay for random 3-SAT, PDSat_PoF's performance is the worst among five alternative algorithms for solving *k*-SAT with long clauses. However, PDSat dramatically outperforms five alternative algorithms on these benchmarks, indicating that Po-PNF and PN-PoF combining PNF and PoF are much more efficient and more effective than PoF itself in the FRW algorithms. Furthermore, PDSat solves 162 instances, 22 instances more than PDSat_nvt does, 19 instances more than PDSat_ntr does, 13 instances more than PDSat_nva does, indicating that the *TBF* mechanism and *Vav* function are effective to improve PD strategy based FRW algorithms. To the best of our knowledge, the PD strategy based on Po-PNF and PN-PoF are currently the only combination strategy that can be used to improve performance of FRW algorithms.

9 Conclusions and future work

We proposed three completely new PD strategies for variable selection based on different probability functions, namely PNF, Po-PNF and PN-PoF, they all are based on the Boltzmann function, which has been evaluated as a fitting function of the *break* value's distribution in the WalkSATlm during the search process. Compared to the existing PoF based PD strategy which loses power on random *k*-SAT instances with long clauses, combining PNF, Po-PNF and PN-PoF has shown its efficiency on random *k*-SAT instances with long clauses.

The main results are summarized below:

1) Based on the WalkSATlm algorithm, we found the distribution of the *break* value and the utilization rate of *break* value tends to be a Boltzmann function.

2) We proposed a PNF according to the Boltzmann function, and then we combine the PNF strategy with a new TBF mechanism to design a new variable selection heuristic called PNF-TBF. It was further combined with the recently proposed *Vav* function led to a new FRW algorithm dubbed PNFSat, which has shown great efficiency and robustness on huge random 7-SAT instances.

3) We did further analyses for PNFSat and found the new tie-breaking strategy which is not suitable for solving medium 7-SAT instances at the threshold ratio of the solubility phase transition, but the alternation version PNFSat_alt significantly outperformed ProbSAT and Score_2SAT on such instances.

4) In order to handle the exponential delay of PNF strategy, we proposed two new heuristics called PN-PoF and Po-PNF respectively, while the parameters were tuned by the Boltzmann function. Combined use of PN-PoF and Po-PNF led to a new FRW algorithm dubbed PN&PoFSat. PN&PoFSat achieved state-of-the-art performance on a broad range of random 5-SAT instances with various variables and ratios.

5) We did further analyses for PN&PoFSat. Sixthly, PNFSat, PNFSat_alt and PN&PoFSat is combined to form flexible new FRW algorithm called PDSat, which consistently outperformed all competitors including state-of-the-art FRW algorithms and two-mode SLS algorithms on solving medium and huge random *k*-SAT instances with long clauses and various ratios as well as sizes in terms of success runs.

6) Besides the efficiency, the experiments also demonstrated that PNF is very robust on random *k*-SAT instances with long clauses, as PNF can be applied to designing efficient FRW algorithms, and cooperates well with several different strategies, such as *Vav* function, *TBF* mechanism and PoF.

For future work, we plan to combine the PNF, Po-PNF, PN-PoF strategy with other algorithmic techniques, such as linear make [13] and configuration checking [3], [14]. Also, inspired by the success of PNF based on the fitting function namely Boltzmann function, we would like to explore the fitting function among configuration checking and other forbidden strategies, and thus combine them to develop more efficient SLS algorithms for random SAT. Additionally, we would like to apply the PNF, Po-PNF and PN-PoF strategies to improving performance of SLS algorithms on solving the structured instances in SAT competition.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Grant No.61673320) and the Fundamental Research Funds for the Central Universities (Grant No.2682019ZT16), and the Fundamental Research Funds for the Central Universities (Grant No.2682020CX59). The authors also gratefully acknowledge the helpful comments and suggestions of the teachers and students from System Credibility Automatic Verification Engineering Lab of Sichuan Province, Southwest Jiaotong University in China, which have improved the presentation and quality.

References

[1] H. Fu, Y. Xu, G. Wu and X. Ning, "An improved genetic algorithm for solving 3-SAT problems based on effective restart and greedy strategy," *in Proc*. 12*th Intelligent Systems and Knowledge Engineering* (*ISKE*), Nanjing, Nov. 2017, pp. 1-6.

[2] H. Fu, Y. Xu, G. Wu, H. Jia, W. Zhang and R. Hu, "An improved adaptive genetic algorithm for solving 3-SAT problems based on effective restart and greedy strategy," *Inter. J. Com. Intell. Sys.*, vol. 11, no. 1, pp.402-413, Jan. 2018.

[3] C. Luo, S. Cai, K. Su and W. Wu, "Clause states based configuration checking in local search for satisfiability," *IEEE Trans. Cybernetic*, vol. 45, no. 5, pp. 1028-1041, May, 2015.

[4] M. J. Heule, "Generating the uniform random benchmarks," *in Proc. SAT competition* 2018, pp. 80.

[5] M. Davis, and H. Putnam. "A computing procedure for quantification theory," *J. ACM*, vol.7, no. 3, pp: 201-215, Jul. 1960.

[6] M. Davis, G. Logemann, and D. W. Loveland, "A machine program for theorem-proving," Commun. ACM, vol. 5, no. 7, pp. 394–397, 1962.

[7] C. Weidenbach, D. Dimov, A. Fietzke, et al, "Wischnewski P. SPASS Version 3.5," *in Proc. on Automated Deduction*, Springer, Berlin, Heidelberg, Aug. 2009, pp. 140-145.

[8] H. H. Hoos and T. Stützle, Stochastic Local Search: Foundations & Applications. San Francisco, CA, USA: Elsevier/Morgan Kaufmann, Sep. 2004.

[9] M. Mavrovouniotis, F.M. Müller and S. Yang, "Ant colony optimization with local search for dynamic traveling salesman problems," *IEEE Trans. Cybernetic*, vol. 47, no. 7, pp. 1743-1756, Jul. 2017.

[10] A. Biere, "CADICAL, LINGELING, PLINGELING, TREENGELING and YALSAT: Solver description," *in Proc. SAT Competition*, 2017: 14-15.

[11] C. Luo, S. Cai, W. Wu, et al, "CSCCSat2014: Solver description," *in Proc. SAT Challenge* 2014, pp. 25–26.

[12] C. Luo, S. Cai, W. Wu, and K. Su, "Focused random walk with configuration checking and break minimum for satisfiability," *in Proc. on Principles and Practice of Constraint Programming*, Springer, Berlin, Heidelberg, Sep. 2013, pp. 481-496.

[13] S. Cai, K. Su, and C. Luo, "Improving walksat for random k-satisfiability problem with k> 3," *in Proc.* 27*th AAAI Conf. on Artif. Intell*, Jun. 2013, pp. 145-151.

[14] S., Cai, and K., Su., "Local search for Boolean satisfiability with configuration checking and subscore". *Artif. Intell*., 2013, 204: 75–98.

[15] A. Balint and A. Fröhlich, "Improving stochastic local search for SAT with a new probability distribution," *in Proc. SAT*, Edinburgh, U.K., Jul. 2010, pp. 10–15.

[16] A. Balint and U. Schöning, "Choosing probability distributions for stochastic local search and the role of make versus break," *in Proc. SAT*, Trento, Italy, Jun. 2012, pp. 16–29.

[17] S. Liu and A. Papakonstantinou. "Local search for hard sat formulas: the strength of the polynomial law," *in* 30*th AAAI Conf. Artif. Intell.*, Feb. 2016, pp. 732-738

[18] A. Balint and U. Schöning, "ProbSAT: Solver description," *in proc. SAT*-2018, pp. 35.

[19] S. Cai and C. Luo, "Score$_2$SAT: Solver description," *in: Pro. SAT*-2017, pp. 34.

[20] H. Fu, W. Zhang, J. Liu, et al, "Improving Stochastic Local Search for SAT by Generating Appropriate Initial Assignment," unpublished.

[21] C. Luo, S. Cai, W. Wu, and K. Su, "Double configuration checking in stochastic local search for satisfiability," *in Proc. Amer. Assoc. Artif. Intell.*, 2014, pp. 2703–2709.

[22] D. A. D. Tompkins, A. Balint, and H. H. Hoos, "Captain Jack: New variable selection heuristics in local search for SAT," in Proc. SAT, Ann Arbor, MI, USA, Jun. 2011, pp. 302–316.

[23] S. Cai, C. Luo, and K. Su. "Improving WalkSAT By Effective Tie-Breaking and Efficient Implementation." Computer Journal (2015).

[24] H. H. Hoos and T. Stützle, "Local search algorithms for SAT: An empirical evaluation," *J. Autom. Reasoning*, 24(4), pp. 421–481, 2000.

[25] D.A. Tompkins and H.H. Hoos, "Dynamic scoring functions with variable expressions: New SLS methods for solving SAT," *in Pro. SAT*-2010, July 2010, pp. 278-292.

[26] C. M. Li and W. Q. Huang, "Diversification and determinism in local search for satisfiability," in *Proc. SAT*, St. Andrews, U.K., Jun. 2005, pp. 158–172.

[27] Ian P. Gent and W. Toby, "Towards an understanding of hill-climbi- ng procedures for SAT," *in Proc. AAAI*, vol. 93, July 1993, pp. 28-33.

[28] S. Cai and K. Su, "Local search for Boolean satisfiability with configuration checking and subscore," *Artif. Intell*., vol. 204, pp. 75–98, Nov. 2013.

[29] S. Cai and K. Su, "Local search with configuration checking for SAT," *in Proc.* 23*rd IEEE Int. Conf. Tools Artif. Intell.* (*ICTAI*), Boca Raton, FL, USA, Nov. 2011, pp. 59–66.

[30] S. Cai and K. Su, "Configuration checking with aspiration in local search for SAT," *in Proc. Amer. Assoc. Artif. Intell.*, 2012, pp. 434–440.

[31] H. Fu, Y. Xu, X. He and X. Ning, "GSAT Algorithm Based on Task Allocation and Scheduling for 3-SAT Problem," *Chinese Journal of Computer engineering& Science*, vol.40, no.8, 1366-1374, Oct. 2018.

[32] S. Cai, C. Luo and K. Su, "Scoring functions based on second level score for k-SAT with long clausess." *Jour. Artif. Intell. Resea.*, vol. 51, no. 2014, pp. 413-441.

[33] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential modelbased optimization for general algorithm configuration," in Learning and Intelligent Optimization. Springer, 2011, pp. 507–523.

[34] Comprehensive score: Towards efficient local search for SAT with long clausess. *In Proc. of IJCAI*, pp, 489-495.

[35] D. Achlioptas,. Random satisfiability. In Handbook of Satisfiability, 2009, pp. 245–270.

[36] B. Selman, H. . Kautz, & B. Cohen, "Noise strategies for improving local search". *In Proc. of AAAI*-94, 1994, pp. 337–343.

[37] A. Braunstein, M. Mézard & R. Zecchina. "Survey propagation: an algorithm for satisfiability". *Random Struct. Algorithms*, 2005, 27(2), pp. 201–226.